COMPOSITIONAL BOUNDED REACHABILITY USING TIME PARTITIONING AND ABSTRACTION

BY

JEREMY D. GREEN

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2012

Urbana, Illinois

Adviser:

Assistant Professor Sayan Mitra

ABSTRACT

Automatic verification of cyber-physical systems (CPS) typically involves computing the reachable set of states of such systems. This computation is known to be exponential in the number of continuous variables. For systems that can be decomposed into separate components with lower dimensionality, we present an algorithm that verifies global safety properties of the complete system using the reach sets of the components. Here, the components are only coupled through a shared time variable. Using a satellite system case study, we are able to show significant savings in memory and runtime computation costs for this approach. For systems whose components are coupled through additional continuous variables, we present an abstraction to overapproximate the interaction between the components such that the aforementioned algorithm can be used. The feasibility of this abstraction is demonstrated experimentally, which also shows additional work is necessary to develop a more efficient abstraction.

To my parents, Ann and Gary Green, for their love and support

ACKNOWLEDGMENTS

I would like to thank my advisor and mentor, Professor Sayan Mitra. His active role in helping to solve my research problems was invaluable. Through his guidance, I've learned how to be mathematically precise in my work and how to approach and explore open ended problems. His passion for hybrid systems research inspired me to pursue this area and complete the work presented in this thesis.

My research group - Taylor Johnson, Zhenqi Huang, Sridhar Duggirala, Adam Zimmerman, and others - were good friends who were always available to give feedback and support in my research. Being able to exchange ideas and discuss difficult problems with them provided some very beneficial collaboration.

Part of this research that involved the satellite system case studies was completed during the Space Scholars summer program at the Air Force Research Laboratory (AFRL) in Albuquerque. I would like to thank my mentor during that summer, Dr. Scott Erwin, and colleague, Rachel Dudley. Scott and Rachel were excellent collaborators, and their expertise and support helped us overcome several technical difficulties in our work.

This research was partially funded by grants from John Deere, National Science Foundation Grant CNS 10-16791, and the AFRL Space Scholars program.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

Computing systems that control physical processes have become commonplace in safety-critical applications such as vehicle control and industrial automation. Design and operation of these *cyber-physical systems (CPS)* should provide a high level of assurance because failures and misbehaviors can lead to significant financial, environmental, and societal losses. A notable recent example was the recall of around 400,000 Prius hybrid cars in 2010 that had a faulty antilock braking system which failed under certain road conditions [1]. Another design and operational mishap occurred on February 10, 2009, when the Iridium 33, a U.S. operational communications satellite, and Cosmos 2251, a Russian decommissioned communications satellite, collided [2]. Aside from the financial costs, the collision created large amounts of space debris that will remain in low earth orbit for several years, posing additional hazards to other satellites.

Although high assurance is a desirable goal, as the systems become more sophisticated, it is becoming more and more challenging to attain using standard design methodologies. The standard technique for detecting design flaws is based on testing and simulations. Since the correctness of these systems crucially depends on the complex interactions between the computing and physical elements, it is difficult and in some cases it is impossible to cover the set of all possible behaviors with a finite number of simulations or tests. Furthermore, the simulation tools often lack precise semantics. This makes it impossible to arrive at rigorous assurance guarantees from the simulation runs.

Formal verification provides an alternative to these approaches. This approach involves first building a mathematical model for the system (with precise semantics), and then using deductive or algorithmic techniques to establish properties about *all* possible behaviors of the model in one go. The most common algorithmic approach involves computing the set of reachable

states of the model. This enables one to check if any of those reachable states are unsafe or "bad". Algorithmic verification has seen success in establishing correctness of many hardware and software systems, and based on recent developments, they present an attractive approach for verification of embedded control systems. Consider the satellite example where we may wish to determine, up to some bounded time, whether there could be a collision with another object or satellite. If all reachable states of our satellite can be calculated up to this time bound, then we can check if this reachable set intersects an unsafe set that corresponds to a potential collision. The key to this approach is being able to compute the set of reachable states.

Over the past two decades, there has been tremendous progress in developing algorithms and data-structures for solving the reachability problem for *hybrid models* which involve both discrete and continuous dynamics [3, 4, 5, 6, 7, 8, 9]. There are several software tools [10, 11, 12, 13, 14, 15] which have been effectively used to analyze complex control systems [16, 17, 18, 19, 20]. Yet, in a system with many interacting components, such as the embedded control system in a car or a satellite, as the number of interacting components increases, computing the reach set quickly becomes intractable. In this thesis, we propose an approach that exploits the modular structure of the models for efficient bounded safety verification.

1.1 Approach Overview

We consider systems built from simpler modules or *components*. Our approach is based on first approximating the influence of one component on another using the reach set computations of the individual components. These computations are relatively inexpensive because they involve components with fewer continuous variables. We have developed an algorithm that successively computes overapproximations of the overall system reach set using the component reach sets while simultaneously doing checks for safety.

The verification algorithm relates the component reach sets through a shared time variable. For a specified time interval, the algorithm creates a conservative overapproximation of the composed system's reach set using the component reach sets. If the overapproximation is sufficient to make a safety decision with respect to the unsafe set, then the algorithm stops. Oth-

erwise, the original time interval is split into two smaller intervals, and safety is checked recursively over these smaller intervals. The overapproximations of the composed reach set become more accurate when created over shorter time intervals.

In extreme cases, for example, in the system with two orbiting satellites, the components (individual satellites) are only coupled through shared time and the reach sets of the components can be computed directly. In general, to compute the component reach sets independently where the subsystems interact through shared variables, we have to somehow abstract the influence of one component on another. For systems whose components are coupled via shared variables without feedback loops, we overapproximate their interaction to be able to compute the component reach sets separately. For any two components in this coupled system there will be one component whose states' evolution is a function of the input signal from the other component. We do the overapproximation by first partitioning the time bound over which the reach sets will be computed into a set of smaller intervals. For the dependent automaton in the system we create an abstraction using this time partition. Over each time interval of the partition in the abstraction, we use the reach set of the non-dependent component to create upper and lower bounds on the range of values the input signal can take over each time interval. We establish a theoretical bound on the approximation error in this abstraction.

We have experimentally evaluated the performance of these approaches by implementing the algorithms in prototype software tools and applying them to several case studies. The feasibility of the abstraction for interacting automata components is demonstrated through a set of case studies. Preliminary experiments show that while improvement in the runtime and memory usage is achieved in some test cases, the additional discrete locations introduced by the abstraction can also negatively affect the computation costs in others. These results suggest the need for a more careful implementation of the time-partitioning algorithm and also a more rigorous performance analysis of this approach with respect to the type of coupling. For loosely coupled systems, algorithms presented in Chapter 3 showed clear improvements in verifying collision avoidance of several 2-satellite systems over direct reachability analysis; in some cases this algorithm beats direct reachability-based algorithm by several (2-3) orders of magnitude.

1.2 Related Work

There is a large and growing body of work on automatic safety verification of cyber-physical systems modeled as different types of hybrid systems. Initially several classes of hybrid systems were identified for which the exact reachability problem is decidable. These classes include timed automata [21], rectangular initialized hybrid automata [4, 3], o-minimal systems [5], and STORMED systems [22]. As these classes have restricted expressive power, subsequently, there has been more focus on practical algorithms and data structures for approximating the reach sets of more general hybrid systems. Several useful types of data-structures have been proposed including polyhedra [23], zonotopes [24], ellipsoids [25] and semi-algebraic sets [26]. Approximate reachability algorithms based on these data-structures are embodied in tools such as ddt [13], HyTech [10], Checkmate [27], PHAVer [11], and more recently SpaceEx [28].

For nonlinear hybrid systems a standard approach is to use hybridization where the state space for a nonlinear system is broken up into a partition of different zones. Here, the nonlinear dynamics within each zone are overapproximated with rectangular, linear, or affine dynamics [29, 30]. We use this approach in Chapter 4 for the satellite case study where the nonlinear dynamics are overapproximated with rectangular dynamics. Hybridization also inspired the abstraction in Chapter 5 where the partitioning is over time intervals instead of the state-space. An alternative approach for handling nonlinear dynamics, that does not involve hybridization, has been presented in [31].

Our approach for decomposition is similar in spirit to the work presented in [32], in which the authors consider affine continuous dynamical systems where subsystems can be separated using what is called an ϵ-decomposition of the matrices defining the differential equations. The reachable states of the subsystems are computed and used to calculate an overapproximation of the composed system. Bounds on the approximation error are calculated for the decomposed system which are then used in the conservative overapproximation of the composed reach set. The efficiency of the approach is dependent on how strongly the subsystems are coupled. In [33], a decompositional approach is also proposed where they consider continuous linear time invariant (LTI) dynamical systems. Given a LTI system, a series of trans-

formations are used to decompose the system into unidirectionally weakly coupled subsystems. Reachability can then be performed on the lower dimension subsystems. The level of conservatism in the reach set is dependent on the how strongly the subsystems are coupled.

These methods differ in that they are exclusively concerned with the reachability computation itself and apply to only linear and affine systems. In our method, on the other hand, the unsafe set for a particular verification problem is used when computing reach set overapproximations of the composed system. The approximations are computed selectively over only the relevant durations of time that are necessary to make a decision about safety with respect to the unsafe set.

1.3 Organization

First, Chapter 2 presents necessary background information and the hybrid automaton model that is used throughout the thesis. Chapter 3 develops the verification algorithms that verify safety properties using the component reach sets of a decomposed system. Here, the components can only be coupled through a shared time axis. In Chapter 4 we present a two-satellite system case study where the verification algorithm is implemented. Then, in Chapter 5 we develop an abstraction to extend the verification technique to systems whose components are coupled through shared continuous variables. The abstraction overapproximates the interactions between the components such that the abstracted component reach sets can be computed independently. The feasibility of this abstraction is demonstrated in Chapter 6 with a numerical example. Finally, Chapter 7 summarizes the results and gives directions for future work.

CHAPTER 2

BACKGROUND

2.1 Functions, Sets, and Variables

We will introduce the hybrid I/O automaton modeling framework, but first we begin with some notation for functions and sets. For any function f we write the domain and range of f as $dom(f)$ and $range(f)$. For a function f and a set $S \subseteq dom(f)$, we write the restriction of f to S as $f \lceil S$. For a function f whose range is a set of functions, we write $f \downarrow S$ for the function g with $dom(g) = dom(f)$ such that $g(c) = f(c) \lceil S \ \forall c \in dom(g)$.

A *variable* is a name used to identify state components of automata and communication channels between automata. Each variable v is associated with a type, $type(v)$, which is the set of values v can take. A *valuation* for a set of variables V, maps each $v \in V$ with a value in $type(v)$. For a set of variables V, $val(V)$ denotes the set of all possible valuations of V. Valuations are denoted by $\mathbf{v}, \mathbf{x}, \mathbf{x}'$, etc. For a valuation $\mathbf{v}$ of V the value of a variable $v \in V$, is denoted by $\mathbf{v}.v \triangleq \mathbf{v} \lceil \{v\}$.

For $\mathbf{x} \in \mathbb{R}^n$, $\gamma \geq 0$, $B_\gamma(\mathbf{x}) \triangleq \{\mathbf{y} : |\mathbf{y} - \mathbf{x}| \leq \gamma\}$ denotes the ball centered at $\mathbf{x}$ of radius γ. The bounded and compact set $S \subseteq \mathbb{R}^n$ expanded by γ is defined as $\mathsf{Expand}(S, \gamma) \triangleq \underset{\mathbf{x} \in S}{\cup} B_\gamma(\mathbf{x})$. Letting ∂S denote the boundary of S, we define $\mathsf{Shrink}(S, \gamma) \triangleq \{\mathbf{x} : \mathbf{x} \in S \wedge (\min_{\mathbf{y} \in \partial S} |\mathbf{y} - \mathbf{x}|) \geq \gamma\}$ as the set S shrunk by γ.

A *trajectory* τ for V is a function $\tau : [0, t] \to val(V)$, where $t \in \mathbb{R}_{\geq 0}$. That is, a trajectory maps an interval $[0, t]$ of time to valuations of V. Example trajectories are shown in Fig. 2.1. A variable is *continuous* if all its trajectories are piecewise-continuous. A *discrete* variable is a special type of continuous variable whose trajectories are piece-wise constant. We define $\tau.\mathsf{fstate} \triangleq \tau(0)$ and $\tau.\mathsf{lstate} \triangleq \tau(t)$. Given a trajectory τ of V, the restriction of the trajectory to a variable $v \in V$ is denoted by $\tau \downarrow v$. The concatenation

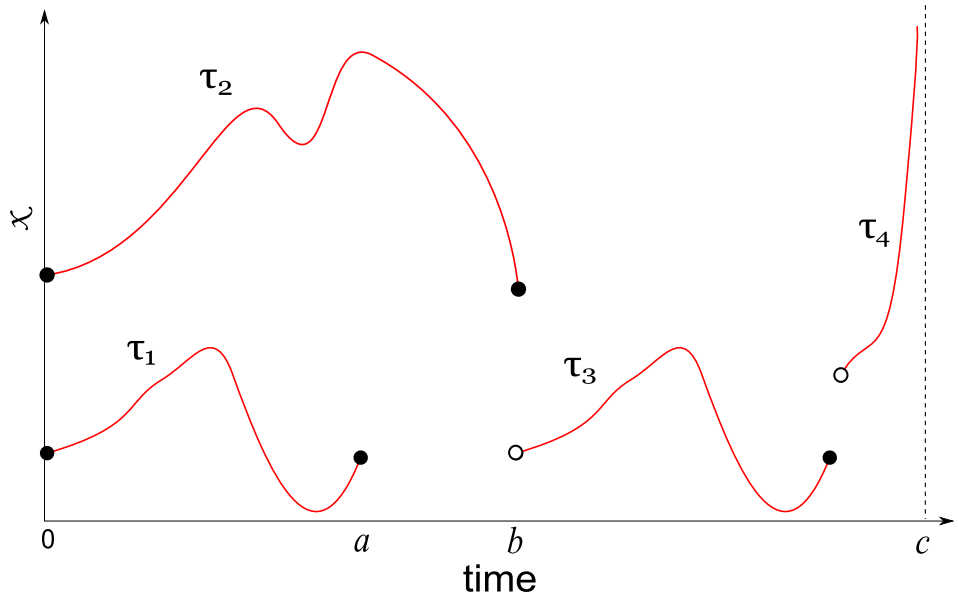

Figure 2.1: Example trajectories for some real-valued continuous variable x.

of two trajectories τ_1, τ_2 is done by taking the union between the first trajectory and the second where the second trajectory has its domain shifted by the limit time of the first as follows: $\tau_1 \frown \tau_2 \triangleq \tau_1 \cup (\tau_2 \lceil (0, \infty) + \tau_1.\mathsf{ltime})$.

A *hybrid sequence* is used to model a combination of instantaneous changes and changes that evolve continuously over a period of time. Using a set of *actions* A to model the instantaneous changes, an $(A, V) - sequence$ is an alternating sequence $\alpha = \tau_0 a_1 \tau_1 a_2 \tau_2 ...$ where V is a set of variables, each τ_i is a trajectory of the variables in V, and each $a_i \in A$.

2.2 Hybrid I/O Automata

The following definition of hybrid I/O automata makes a few minor changes to the standard one [34, 35] for a cleaner presentation of the results in this paper.

Definition 1. *The* hybrid I/O automaton (HIOA) *is a tuple* $\mathcal{A} \triangleq \langle V, L, \Theta, A, \mathcal{D}, \mathcal{T} \rangle$, *where:*

(a) V *is a set of variables.* V *consists of a single discrete variable* loc *of type* L*, and sets of (continuous) internal (*X*), input (*U*), and output (*Y*) variables. The set* $Q \triangleq val(\{loc\} \cup X)$ *is called the set of* states.

(b) $\Theta \subseteq Q$ *is a set of* initial states.

(c) A *is a set of (internal)* transition labels.

(d) $\mathcal{D} \subseteq Q \times A \times Q$ *is a set of discrete state transitions. For a transition* $((\ell, \mathbf{x}), a, (\ell', \mathbf{x}')) \in \mathcal{D}$ *we write* $(\ell, \mathbf{x}) \xrightarrow{a}_{\mathcal{A}} (\ell', \mathbf{x}')$ *in short and drop the suffix* $\mathcal{A}$ *when the automaton is clear from context.*

(e) $\mathcal{T}$ *: Set of trajectories for* V *that is closed under prefix, suffix, and concatenation (see [34, 35] for details).*

In addition, a HIOA must satisfy the input enabled condition: For any state $(\ell, \mathbf{x}) \in Q$ *and any time-bounded trajectory* ν *of* U*, there exists* τ *with* $\tau.\mathsf{fstate} = (\ell, \mathbf{x})$ *such that either (a)* $(\tau \downarrow U) = \nu$ *or (b)* $(\tau \downarrow U)$ *is a prefix of* ν *and some discrete transition is enabled at* $\tau.\mathsf{lstate}$*.*

Typically $\mathcal{D}$ is syntactically specified by *guards* and *reset maps* for each $a \in A$. For any $a \in A$, $Grd(a) \triangleq \{(\ell, \mathbf{x}) \mid \exists\ (\ell', \mathbf{x}')$ such that $(\ell, \mathbf{x}) \xrightarrow{a} (\ell', \mathbf{x}')\}$. We define the set-valued function $Rst(a)$ which maps each $(\ell, \mathbf{x}) \in Grd(a)$ to the set $\{(\ell', \mathbf{x}') \mid$ such that $(\ell, \mathbf{x}) \xrightarrow{a} (\ell', \mathbf{x}')\}$. Typically $\mathcal{T}$ is syntactically specified by invariants, stopping conditions, and differential equations. For each $\ell \in L$, the corresponding invariant $Inv(\ell)$ and stopping condition $Stop(\ell)$ are subsets of Q, and $Flow(\ell)$ is a set of differential and algebraic equations (or inequalities) involving the continuous variables $X \cup U \cup Y$. A trajectory τ for V is in the set $\mathcal{T}_\ell$ iff (a) $(\tau \downarrow loc)(0) = \ell$, (b) $\tau \downarrow V$ is a solution of $flow(\ell)$ for the given trajectory $(\tau \downarrow U)$ of the input variables, (c) For any $t \in dom(\tau)$, $(\tau \downarrow \{loc \cup X\})(t) \in Inv(\ell)$, and (d) For any $t \in dom(\tau)$, if $(\tau \downarrow \{loc \cup X\})(t) \in Stop(\ell)$ then $t = \tau.\mathsf{ltime}$. The set of trajectories $\mathcal{T}$ is the union $\cup_{\ell \in L} \mathcal{T}_\ell$. See [35] for details and examples. For a variable $x \in X, u \in U$, the differential equation $\dot{x} = f(x, u)$ specified trajectories which are solutions of this equation for any trajectory of u. For an automaton $\mathcal{A}$, we refer to the components of the automaton as $V_\mathcal{A}, X_\mathcal{A}, Q_\mathcal{A}, \Theta_\mathcal{A}$, $A_\mathcal{A}$ etc. For automaton $\mathcal{A}_i$, the components will be denoted by V_i, X_i, Q_i, Θ_i, etc.

Next, we define the *parallel composition* operation on HIOAs which is used for constructing larger models from two interacting models. In this paper, we

consider HIOA-components that only interact through shared input/output variables[1]

Definition 2. *Hybrid automata* $\mathcal{A}_1$ *and* $\mathcal{A}_2$ *are* compatible *if* $X_1 \cap V_2 = X_2 \cap V_1 = \emptyset$. *If* $\mathcal{A}_1$ *and* $\mathcal{A}_2$ *are compatible, then their composition* $\mathcal{A}_1 \| \mathcal{A}_2$ *is defined as* $\mathcal{A} \triangleq \langle V, L, \Theta, A, \mathcal{D}, \mathcal{T} \rangle$, *where:*

(a) $X = X_1 \cup X_2$, $Y = Y_1 \cup Y_2$, $U = U_1 \cup U_2 \setminus (Y_1 \cup Y_2)$, *and* $V = \{loc\} \cup X \cup Y \cup U$, *where loc has type* $L = L_1 \times L_2$. *We denote the first and the second components of a* $\ell \in L$ *by* ℓ_1 *and* ℓ_2.

(b) $\Theta = \{(\ell, \mathbf{x}) \in Q | (\ell_1, \mathbf{x}\lceil X_1) \in \Theta_1 \wedge (\ell_2, \mathbf{x}\lceil X_2) \in \Theta_2\}$.

(c) $A = A_1 \cup A_2$.

(d) $\mathcal{D} \subset Q \times A \times Q$ *For each* $\mathbf{x}, \mathbf{x}' \in Q$ *and each* $a \in A$, $\mathbf{x} \xrightarrow{a}_{\mathcal{A}} \mathbf{x}'$ *iff for* $i = 1, 2$, *either* (1) $a \in A_i$ *and* $\mathbf{x}\lceil X_i \xrightarrow{a}_i \mathbf{x}'\lceil X_i$ *or* (2) $a \notin A_i$ *and* $\mathbf{x}\lceil X_i = \mathbf{x}'\lceil X_i$.

(e) $\mathcal{T} \subset trajs(V)$ *is defined by* $\tau \in \mathcal{T} \leftrightarrow (\tau \downarrow V_i) \in \mathcal{T}_i$, $i \in \{1, 2\}$.

2.3 Executions, Reach Sets, and Safety

An *execution fragment* α of automaton $\mathcal{A}$ is a (A, V)-sequence $\alpha = \tau_0 a_1 \tau_1 a_2 ...$, where each $\tau_i \in \mathcal{T}, a_i \in A$, and $\tau_{i-1}.\mathsf{lstate} \xrightarrow{a_i} \tau_i.\mathsf{fstate}$. The first state of α, $\alpha.\mathsf{fstate}$ is denoted by $\tau_0.\mathsf{fstate}$. If α is a finite sequence ending with a closed trajectory τ_n, then its last state $\alpha.\mathsf{lstate}$ is defined as $\tau_n.\mathsf{lstate}$ and its duration $\alpha.\mathsf{ltime}$ is defined as $\sum_{i=0}^{n} \tau.\mathsf{ltime}$. Along execution α, $\mathcal{A}$'s state may not be uniquely defined at a given time $t \in [0, \alpha.\mathsf{ltime}]$: one or more discrete transitions occur at time t. We adopt the convention that the state at time t is the unique state after all possible transitions at time t. Formally, for any time $t \in [0, \alpha.\mathsf{ltime}]$, we define the state of $\mathcal{A}$ at time t, denoted by $\alpha(t)$, as $\alpha'.\mathsf{lstate}$ where α' is the longest prefix of α with $\alpha'.\mathsf{ltime} \leq t$. An execution fragment is an *execution* if it also starts at an initial state, that is, $\alpha.\mathsf{fstate} \in \Theta$. The set of all executions and execution fragments are denoted by $\mathsf{Execs}_{\mathcal{A}}$ and $\mathsf{Frags}_{\mathcal{A}}$. The set of executions and execution fragments up to time T are denoted $\mathsf{Execs}^T_{\mathcal{A}}$ and $\mathsf{Frags}^T_{\mathcal{A}}$.

[1]This is our HIOA definition has only internal transitions.

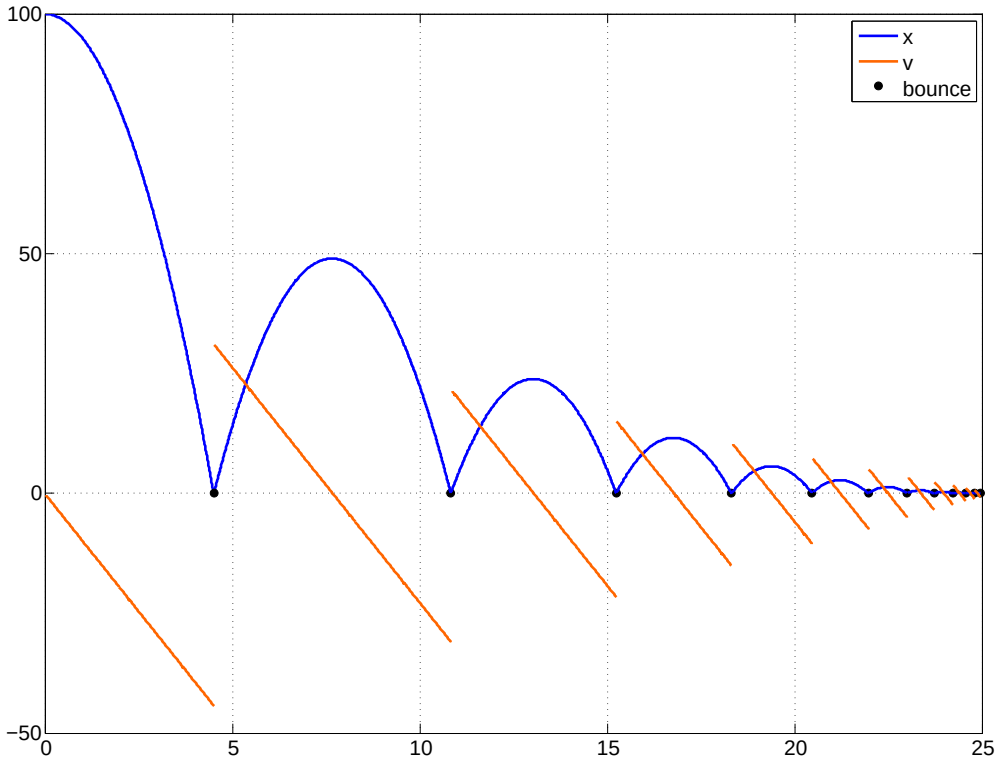

Figure 2.2: Bouncing ball example execution.

An example execution from a bouncing ball automaton is shown in Fig. 2.2. The position x and velocity v evolve according to the differential equations $\frac{d}{dt}(x) = v$ and $\frac{d}{dt}(v) = -g$ where g is the acceleration due to gravity. Each bounce is modeled as discrete jump. The guard for the bounce transition is $\{x = 0 \wedge v < 0\}$, and the resets are $x' = x$ and $v' = -\rho * v$. Here, ρ is a scaling factor that models the energy dissipation from each bounce, and $0 < \rho < 1$.

A state $(\ell, \mathbf{x})$ is *reachable* if there exists an execution α with $\alpha.\mathsf{lstate} = (\ell, \mathbf{x})$. Define $\mathsf{Reach}_{\mathcal{A}}(t_1, t_2)$ as $(\ell, \mathbf{x}) \in \mathsf{Reach}_{\mathcal{A}}(t_1, t_2)$ iff there exists an execution $\alpha \in \mathsf{Execs}_{\mathcal{A}}$ and a time $t \in [t1, t2]$ such that $\alpha(t) = \mathbf{x}$. We write $\mathsf{Reach}_{\mathcal{A}}(t, t)$ simply as $\mathsf{Reach}_{\mathcal{A}}(t)$ and $\mathsf{Reach}_{\mathcal{A}}(0, T)$ as $\mathsf{Reach}_{\mathcal{A}}^{T}$. Given a set of states $\mathcal{U} \subseteq Q$, $\mathcal{A}$ is said to be T-*safe* with respect to $\mathcal{U}$ if $\mathsf{Reach}_{\mathcal{A}}^{T}$ and $\mathcal{U}$ are disjoint. It is said to be *safe* if it is T-safe for all T. Otherwise it is said to be *unsafe*.

Several special classes of hybrid automata have been identified for which $\mathsf{Reach}_{\mathcal{A}}$ and $\mathsf{Reach}_{\mathcal{A}}^{T}$ can be computed exactly [21, 36, 5, 22]. For general

automata with complex continuous dynamics, exact computation is undecidable and one has to rely on overapproximations for safety verification. Over the past decade, several algorithms and tools have been developed for computing overapproximations of bounded reach sets of different classes of hybrid automata. Examples include tools such as HyTech [10], UPPAAL [12], PHAVer [11], and the more recent SpaceEx [28]. In this paper, we will make use of these reach set overapproximations. Overapproximations of $\mathsf{Reach}_{\mathcal{A}}$ and $\mathsf{Reach}^T_{\mathcal{A}}$ will be denoted by $\overline{\mathsf{Reach}}_{\mathcal{A}}$ and $\overline{\mathsf{Reach}}^T_{\mathcal{A}}$. Thus, for a state $(\ell, \mathbf{x}) \in \overline{\mathsf{Reach}}^T_{\mathcal{A}}$, it is not guaranteed that there exists an execution of $\mathcal{A}$ that actually reaches $(\ell, \mathbf{x})$ within time T. We introduce cartesian products of reach set overapproximations that will be used in the subsequent chapters.

Definition 3. *Let $\mathcal{A}_1, \mathcal{A}_2, ..., \mathcal{A}_n$ be a collection of HIOAs. For a $[t_1, t_2]$ interval, we define the following composition overapproximations:*

1. *$C_i(t_1, t_2)$ is a convex polyhedra containing $\overline{\mathsf{Reach}}_{\mathcal{A}_i}(t_1, t_2)$*
2. *$C(t_1, t_2) \triangleq \prod_{i \in [n]} C_i(t_1, t_2)$*

$C_i(t_1, t_2)$ can be computed by taking the convex hull of $\overline{\mathsf{Reach}}_{\mathcal{A}_i}(t_1, t_2)$.

2.4 Hiding Variables and Traces

An execution records information about *all* variables and transitions over a particular run of the system. Sometimes it is useful to consider the information about only a subset of variables and transitions. Formally, this is achieved by restricting the execution. For any $A' \subseteq A$ and $V' \subseteq V$, the (A', V')-restriction of an execution $\alpha = \tau_0 a_1 \tau_1 \dots \tau_n$ is an (A', V')-sequence β which is defined in two steps as follows: (1) Each $a_i \notin A'$ is removed from α. (2) All consecutive trajectories $\tau_i \tau_{i+1}$ are replaced by a single concatenated trajectory $\tau_i \frown \tau_{i+1}$. (3) Each τ_i is replaced by $\tau_i \downarrow V_i$. The resulting $\beta = \tau'_0 a'_1 \tau'_1 \dots \tau'_n$ is a (A', V')-sequence.

Lastly, in Chapter 5 we will be analyzing HIOA that interact via shared variables where the following definition will be necessary.

Definition 4. *Consider two HIOA automata $\mathcal{A}_1$ and $\mathcal{A}_2$. $\mathcal{A}_1$ has a variable dependency on $\mathcal{A}_2$ if $\exists\ x \in X_1$ of $\mathcal{A}_1$, and $\exists y \in Y_2$, where y is mapped to an input variable $u \in U_1$.*

CHAPTER 3

VERIFICATION ALGORITHM

In this chapter we present an algorithm for safety verification of a class of non-interacting HIOAs. The complete system is a HIOA $\mathcal{A}$ which is a composition $\mathcal{A} = \mathcal{A}_0 \| \ldots \| \mathcal{A}_{n-1}$, but none of the component automata have any input variables. The only coupling between these automata is the shared notion of time. Automata compositions with variable dependencies will be considered in Chapter 5. Safety is determined with respect to some unsafe set $\mathcal{U}$ defined by valuations of the continuous variables of the component automata. Recall that, for any time bound $T \geq 0$, $\mathcal{A}$ is T-*safe* with respect to $\mathcal{U}$ iff $\mathcal{U} \cap \mathsf{Reach}_{\mathcal{A}}^{T} = \emptyset$. We will not compute $\overline{\mathsf{Reach}}_{\mathcal{A}}^{T}$ directly, but instead, we will use the reach sets of the component automata. Let the automata $\mathcal{A}_i, i \in [n]$ be augmented with a continuous *timer* variable. This timer variable will be used to relate component reach sets. For some $[t_{min}, t_{max}] \subseteq [0, T]$, the component reach sets will be combined according to Definition 3 to create convex polyhedra that overapproximate the reach set of $\mathcal{A}$ over that interval.

3.1 Bounded Safety with Approximate Reach Sets

The verification process is implemented through Verify1 (shown in Algorithm 1) and its subroutine ReachCompCheck1 (Algorithm 2). In Verify1, two global safety flags, UnsafeFlag and UndecidedFlag, are initialized to **false** but can be modified in the subroutine. These flags are used to determine the safety of the entire system. ReachCompCheck1 uses a parameter $\epsilon > 0$ and $\overline{\mathsf{Reach}}_{\mathcal{A}_i}^{T}$ for each $i \in [n]$. Each call to ReachCompCheck1 computes an overapproximation $C(t_1, t_2)$ of the reach set of $\mathcal{A}$ over a time interval $[t_1, t_2]$ by taking the Cartesian product of the convex hull of the $\overline{\mathsf{Reach}}_{\mathcal{A}_i}(t_1, t_2)$'s. A call to ReachCompCheck1 can lead to four possible cases. First, if $C(t_1, t_2)$ is disjoint from $\mathcal{U}$ then the global safety flags remain unchanged. Second,

if $C(t_1, t_2)$ is contained in $\mathcal{U}$ then UnsafeFlag is set to **true** . Otherwise, ReachCompCheck1 is called recursively over two smaller time intervals that equally divide $[t_1, t_2]$, unless the duration of the interval $t_2 - t_1$ is already shorter ϵ in which case UndecidedFlag is set to **true** .

Algorithm 1 Verify1$(T, \mathcal{A}, \mathcal{U}, \epsilon)$

$\forall i \in [1, ..., n]$ compute $\overline{\mathsf{Reach}}^{T}_{\mathcal{A}_i}$
UnsafeFlag = **false**
UndecidedFlag = **false**
ReachCompCheck1$(0, T)$
if UnsafeFlag **then**
  return UNSAFE
else if UndecidedFlag **then**
  return UNDECIDED
else
  return SAFE
end if

Algorithm 2 ReachCompCheck1(t_{min}, t_{max})

$\forall i \in [1, ..., n]$
$C_i(x, t_{min}, t_{max}) \leftarrow convexhull(\overline{\mathsf{Reach}}_{\mathcal{A}_i}(t_{min}, t_{max}))$
if $\mathcal{U} \cap C(t_{min}, t_{max}) = \emptyset$ **then**
  return
else if $C(t_{min}, t_{max}) \subseteq \mathcal{U}$ **then**
  UnsafeFlag = **true**
  return
else if $(t_{max} - t_{min}) < \epsilon$ **then**
  UndecidedFlag = **true**
  return
else
  ReachCompCheck1$(t_{min}, t_{min} + \frac{t_{max} - t_{min}}{2})$
  ReachCompCheck1$(t_{min} + \frac{t_{max} - t_{min}}{2}, t_{max})$
end if

Termination of Verify1 is obvious: for an initial call to ReachCompCheck1 over the interval $[0, T]$, the depth of the recursive call tree is at most $\log \frac{T}{\epsilon}$. When a particular call terminates without making additional recursive calls its denoted as a *leaf node* of the call tree.

If all the leaves' $C(t_{min}, t_{max})$s do not intersect $\mathcal{U}$ then Verify1 decides SAFE, if one of the leaves' $C(t_{min}, t_{max})$ is contained in $\mathcal{U}$ then the verification algorithm decides UNSAFE, otherwise it returns UNDECIDED.

A run of Verify1 is demonstrated in Fig. 3.1 for the two-satellite system that is discussed in detail in Chapter 4. The system is modeled by two automata, each having a continuous timer variable and an angular position variable θ. Shown in the figure are the angular positions of the two satellites. The actual reach set of the composed system is not shown, but it starts from $(0.25, 0)$, travels to the upper right of the state space, and is contained within the blue set. The unsafe set is represented by two small polyhedra that are highlighted by the red circles. The numbers indicate the compositions that are calculated as rectangles for the first four levels of the call tree that was executed (deeper levels of the call tree are not labeled). In the first call to the subroutine ReachCompCheck1, the single lightest gray box labeled "1" is computed. In this figure it is overlapped by other compositions in subsequent calls to ReachCompCheck1. Here, there is a nonempty intersection with the unsafe set, but the intersection is not contained within the unsafe set, so ReachCompCheck1 is called recursively. The time interval is split and the two compositions at the next depth in the call tree are labeled "2". Once a call is made with a composition that has an empty intersection with the unsafe set, it is a terminating call and is colored blue. The process continues with recursive calls to ReachCompCheck1 until, in this case, none of the terminating compositions intersect the unsafe set, and Verify1 can return SAFE.

The following assumption about the reach set computations will be used in establishing Proposition 1. It states that for any $\epsilon > 0$, there exists a bound γ such that the composite overapproximation of the reach set over an interval of length ϵ is contained in the actual computed reach set bloated by γ.

Assumption 3.1.1. *For any $\epsilon > 0$, $T > 0$ there exits a $\gamma > 0$ such that for all $t \in [0, T - \epsilon]$, $C(t, t + \epsilon) \subseteq \mathsf{Expand}(\overline{\mathsf{Reach}}_{\mathcal{A}}(t, t + \epsilon), \gamma)$.*

The following proposition asserts the correctness of Verify1 for bounded safety verification. If it returns SAFE, then the T-bounded reach set does not intersect the unsafe set. If the γ bloated reach set does not intersect the unsafe set, then the algorithm will return SAFE. If UNSAFE is returned, then there exists a time interval greater than or equal to $\frac{\epsilon}{2}$ such that the reach set over that interval is contained in the unsafe set. If there exists a time interval greater than or equal to 2ϵ such that the γ bloated reach set over

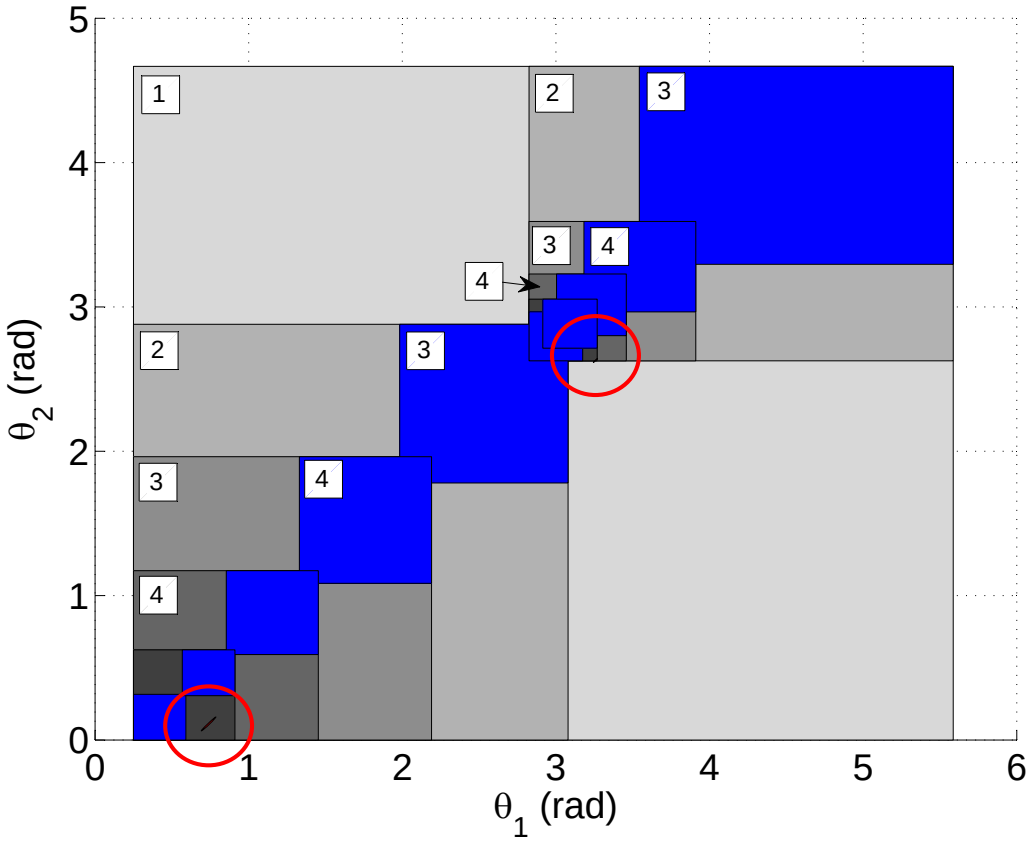

Figure 3.1: Compositions from a run of Verify1 which decided safe. The gray boxes represent compositions which had non-empty intersections with $\mathcal{U}$, but were not contained in $\mathcal{U}$, thus Verify1 was called recursively. The darker the shade the deeper the call to Verify1 in the call tree. The blue set represents the compositions of the terminating calls of Verify1, which did not intersect $\mathcal{U}$. The unsafe set $\mathcal{U}$ is highlighted by the red circles. Parameters: $e_1 = 0.328$, $a_1 = 19025km$, $e_2 = 0.324$, $a_2 = 20184$, $partitionsize = 20deg$,$T = 0.85$ orbits, $d = 100km$, init: $(0.25, 0)$.

that interval is contained in the unsafe set, then the algorithm will decide UNSAFE. If Verify1 returns UNDECIDED, then the γ bloated reach set has a nonempty intersection with the unsafe set, but for any 2ϵ time interval, the γ bloated reach set over that interval is not a subset of the unsafe set. If the reach set has non empty intersection with the unsafe set and for any $\frac{\epsilon}{2}$ time interval the reach set over that interval is not a subset of the unsafe set, then Verify1 will return UNDECIDED.

Proposition 1. *For any time bound $T \geq 0$, and parameter value $\epsilon > 0$, there exists a γ such that the decision made by Verify1 is related to the computed reach set of $\mathcal{A}$ as follows:*

(A0) $SAFE \Rightarrow \overline{\mathsf{Reach}}^{T}_{\mathcal{A}} \cap \mathcal{U} = \emptyset$.

(A1) $\mathsf{Expand}(\overline{\mathsf{Reach}}^{T}_{\mathcal{A}}, \gamma) \cap \mathcal{U} = \emptyset \Rightarrow SAFE$.

(B0) $UNSAFE \Rightarrow \exists\ t, t' \in [0, T]$ *with* $t' - t \geq \frac{\epsilon}{2}$, $\overline{\mathsf{Reach}}_{\mathcal{A}}(t, t') \subseteq \mathcal{U}$.

(B1) $\exists\ t, t' \in [0, T]$ *with* $t' - t \geq 2\epsilon$, *such that* $\mathsf{Expand}(\overline{\mathsf{Reach}}_{\mathcal{A}}(t', t), \gamma) \subset \mathcal{U}$ $\Rightarrow$ *UNSAFE.*

(C0) $UNDECIDED \Rightarrow \mathsf{Expand}(\overline{\mathsf{Reach}}^{T}_{\mathcal{A}}, \gamma) \cap \mathcal{U} \neq \emptyset \wedge \forall t\ \mathsf{Expand}(\overline{\mathsf{Reach}}_{\mathcal{A}}(t, t + 2\epsilon), \gamma) \not\subseteq \mathcal{U}$.

(C1) $\overline{\mathsf{Reach}}^{T}_{\mathcal{A}} \cap \mathcal{U} \neq \emptyset \wedge \forall t\ \overline{\mathsf{Reach}}_{\mathcal{A}}(t, t + \frac{\epsilon}{2}) \not\subseteq \mathcal{U} \Rightarrow UNDECIDED.$

Proof. We fix ϵ and T. Let $\Lambda = \{[t_0, t_1], [t_1, t_2], ..., [t_k, t_{k+1}]\}$ be the set of intervals corresponding to the leaves of the final call tree. Since the interval at each recursive call is split into two, it follows that $\bigcup_{\sigma \in \Lambda} = [0, T]$.
$(A0)$ If the verification algorithm decides SAFE then for each leaf node $\sigma \in \Lambda$, the corresponding $C(\sigma) \cap \mathcal{U} = \emptyset$. From Definition 3 we know that for any time interval σ, $\overline{\mathsf{Reach}}_{\mathcal{A}}(\sigma) \subseteq C(\sigma)$, and therefore, $\overline{\mathsf{Reach}}^{T}_{\mathcal{A}} \subseteq \bigcup_{\sigma \in \Lambda} C(\sigma)$. It follows that $\overline{\mathsf{Reach}}^{T}_{\mathcal{A}} \cap \mathcal{U} = \emptyset$.
$(A1)$ If $\mathsf{Expand}(\overline{\mathsf{Reach}}^{T}_{\mathcal{A}}, \gamma) \cap \mathcal{U} = \emptyset$ then $\forall\ t \in [0, T - \epsilon]$ the composition $C(t, t + \epsilon) \cap \mathcal{U} = \emptyset$ by Assumption 3.1.1. Thus, for all $\sigma \in \Lambda$, $C(\sigma) \cap \mathcal{U} = \emptyset$ and the algorithm returns SAFE.
$(B0)$ If the algorithm decides UNSAFE then $\exists \sigma \in \Lambda$ such that $C(\sigma) \subset \mathcal{U}$. Since $\mathsf{Reach}_{\mathcal{A}}(\sigma) \subset C(\sigma)$, $\mathsf{Reach}_{\mathcal{A}}(\sigma) \subset \mathcal{U}$.
$(B1)$ If there exists $t_1, t_2 \in [0, T]$, where $t_2 - t_1 \geq 2\epsilon$, such that $\mathsf{Expand}(\mathsf{Reach}_{\mathcal{A}}(t_1, t_2), \gamma) \subset \mathcal{U}$, then $\forall t_{min}$, t_{max} where $\frac{\epsilon}{2} \leq t_{max} - t_{min} \leq \epsilon$ and $t_1 \leq t_{min} < t_{max} \leq t_2$, $C(t_{min}, t_{max}) \subseteq \mathcal{U}$ by *Assumption* 3.1.1. The requirement $t_2 - t_1 \geq 2\epsilon$ is necessary since the time partitioning can take place any where along the interval t_1 to t_2. Therefore, there will be at least one $\sigma \in \Lambda$ such that $C(\sigma) \subseteq \mathcal{U}$ and this σ will lead the algorithm to decide UNSAFE.
$(C0)$ If Verify1 terminates with UNDECIDED then we know, first, that for some leaf $\sigma \in \Lambda$ UndecidedFlag was set to **true**. That is, $C(\sigma) \cap \mathcal{U} \neq \emptyset$, where the duration of the interval σ is between $\frac{\epsilon}{2}$ and ϵ. From Assumption 3.1.1, $C(\sigma) \subseteq \mathsf{Expand}(\overline{\mathsf{Reach}}_{\mathcal{A}}(\sigma), \gamma)$ and $\mathsf{Expand}(\overline{\mathsf{Reach}}_{\mathcal{A}}(\sigma), \gamma) \cap \mathcal{U} \neq \emptyset$. Secondly, none of the leaves set UnsafeFlag to **false**, that is, for any $\sigma \in \Lambda$ $C(\sigma) \cap \mathcal{U} \not\subseteq \mathcal{U}$. For each σ there are then two possibilities.

(a) First, σ could be safe such that $C(\sigma) \cap \mathcal{U} = \emptyset$. Since $\mathsf{Reach}_{\mathcal{A}}(\sigma) \subseteq C(\sigma)$, $\mathsf{Reach}_{\mathcal{A}}(\sigma) \cap \mathcal{U} = \emptyset$. Then, $\forall t_1, t_2 \in \sigma$, $\mathsf{Expand}(\mathsf{Reach}_{\mathcal{A}}(t_1, t_2), \gamma) \not\subseteq \mathcal{U}$.

(b) Or, σ could set UndecidedFlag to **true** implying that the duration of σ is bounded by $\frac{\epsilon}{2}$ and ϵ and $C(\sigma) \not\subseteq \mathcal{U}$. Using Assumption 3.1.1, $\mathsf{Expand}(\mathsf{Reach}_{\mathcal{A}}(\sigma), \gamma) \not\subseteq \mathcal{U}$.

Therefore, either any $[t, t + 2\epsilon]$ time interval intersects a safe time interval $[t_1, t_2]$ where $[t_1, t_2] \subseteq [t, t + 2\epsilon]$ and $\mathsf{Expand}(\mathsf{Reach}_{\mathcal{A}}(t_1, t_2), \gamma) \not\subseteq \mathcal{U}$, or $[t, t + 2\epsilon]$ contains a σ interval that set UndecidedFlag to **true** such that $\mathsf{Expand}(\mathsf{Reach}_{\mathcal{A}}(\sigma), \gamma) \not\subseteq \mathcal{U}$.
$(C1)$ If $\overline{\mathsf{Reach}}^{T}_{\mathcal{A}} \cap \mathcal{U} \neq \emptyset$, then there exists a $\sigma \in \Lambda$, such that $C(\sigma) \cap \mathcal{U} \neq \emptyset$ and σ did not return SAFE. If $\forall t\ \overline{\mathsf{Reach}}_{\mathcal{A}}(t, t+\frac{\epsilon}{2}) \not\subseteq \mathcal{U}$, then $\forall t\ C(t, t+\frac{\epsilon}{2}) \not\subseteq \mathcal{U}$ and thus any σ does not return UNSAFE. The first statement also implies that there exists a leaf that returned UNDECIDED.

□

Algorithm 3 Verify2$(T, \mathcal{A}, \mathcal{U}, \epsilon)$

```
1: ∀i ∈ [1, ..., n] compute Reach^T_{A_i}
2: UnsafeFlag = false
3: PossiblyUnsafeFlag = false
4: UndecidedFlag = false
5: ReachCompCheck2(0, T)
6: if UnsafeFlag then
7:    return UNSAFE
8: else if PossiblyUnsafeFlag then
9:    return POSSIBLY UNSAFE
10: else if UndecidedFlag then
11:    return UNDECIDED
12: else
13:    return SAFE
14: end if
```

3.2 Bounded Safety with Exact Reach Sets

In this section we consider a refinement for the previous algorithm which relies on exact bounded reach set computations. Several classes of HA have

Algorithm 4 ReachCompCheck2(t_{min}, t_{max})

$\forall i \in [1, ..., n]$
$C_i(x, t_{min}, t_{max}) \leftarrow convexhull(\mathsf{Reach}_{\mathcal{A}_i}(t_{min}, t_{max}))$
if $\mathcal{U} \cap C(t_{min}, t_{max}) = \emptyset$ **then**
  return
else if $C(t_{min}, t_{max}) \subseteq \mathcal{U}$ **then**
  UnsafeFlag = **true**
  return
else if $(t_{max} - t_{min}) < \epsilon$ **then**
  if $\mathsf{Shrink}(\mathcal{U}, \gamma) \cap C(t_{min}, t_{max}) \neq \emptyset$ **then**
    PossiblyUnsafeFlag = **true**
    return
  else
    UndecidedFlag = **true**
    return
  end if
else
  ReachCompCheck2($t_{min}, t_{min} + \frac{t_{max} - t_{min}}{2}$)
  ReachCompCheck2($t_{min} + \frac{t_{max} - t_{min}}{2}, t_{max}$)
end if

been identified in the literature which fit this requirement (see for example [21, 36, 5, 22]). If $\mathsf{Reach}_{\mathcal{A}}^{\mathcal{T}}$ can be computed exactly, then ReachCompCheck1 can be augmented with an additional check to gain more information from the UNDECIDED case. The augmented algorithms, Verify2 (shown in Algorithm 3) and ReachCompCheck2 (Algorithm 4), use a new global variable called PossiblyUnsafeFlag, where Verify2 can now return POSSIBLY UNSAFE in addition to the original safety decisions. In the previous algorithms where overapproximated reach sets were used, if $\overline{\mathsf{Reach}}_{\mathcal{A}}^{T}$ was never contained in the unsafe set for some time interval but still intersected it, then we could not say that at least one execution of the real system entered the unsafe set. Now, with exact reach sets, we can distinguish between the undecided case and the possibly unsafe case where at least one execution of the real system enters the unsafe set.

The semantics of the SAFE and UNSAFE decisions remain unchanged. If all the leaves' $C(\sigma)$s are not contained in $\mathcal{U}$, but for one leaf σ $C(\sigma)$ intersects $\mathsf{Shrink}(\mathcal{U}, \gamma)$, then Verify2 decides POSSIBLY UNSAFE. If none the leaves' $C(\sigma)$s are contained in $\mathcal{U}$, but for one leaf σ $C(\sigma)$ has a nonempty intersect with $\mathcal{U}$ and an empty intersection with $\mathsf{Shrink}(\mathcal{U}, \gamma)$, then UNDECIDED is

returned.

If Verify2 returns POSSIBLY UNSAFE, then for all 2ϵ time intervals the γ bloated reach set over that interval is not contained in the unsafe set, but the reach set intersects the unsafe set. If for all $\frac{\epsilon}{2}$ time intervals, the reach set over that interval is not contained in the unsafe set and the reach set intersects the unsafe set shrunk by γ, then the algorithm will decide POSSIBLY UNSAFE. If UNDECIDED is returned then for all 2ϵ time intervals, the γ bloated reach set over those intervals is not contained in the unsafe set and reach set does not intersect the unsafe set shrunk by γ.

There are no conditions on the reach set that will guarantee UNDECIDED is chosen. When undecided is chosen, there exists a $C(\sigma)$ that has a non-empty intersection with $\mathcal{U}$, but has an empty intersection with $\mathsf{Shrink}(\mathcal{U}, \gamma)$. Here, the actual reach set may or may not intersect the unsafe set. It is not suffient to say $\mathsf{Reach}_{\mathcal{A}}^{T} \cap \mathcal{U} \neq \emptyset \wedge \mathsf{Reach}_{\mathcal{A}}^{T} \cap \mathsf{Shrink}(\mathcal{U}, \gamma) = \emptyset$ to guarantee undecided. In this situation, there could still be a $C(\sigma)$ that intersects $\mathsf{Shrink}(\mathcal{U}, \gamma)$.

Proposition 2. *For any time bound $T \geq 0$, and parameter value $\epsilon > 0$, there exists a γ such that the POSSIBLY UNSAFE and UNDECIDED decisions made by Verify2 is related to the computed reach set of $\mathcal{A}$ as follows:*

(A0) *POSSIBLY UNSAFE* $\Rightarrow \forall\, t\ \mathsf{Expand}(\mathsf{Reach}_{\mathcal{A}}(t, t{+}2\epsilon), \gamma) \not\subseteq \mathcal{U} \wedge \mathsf{Reach}_{\mathcal{A}}^{T} \cap \mathcal{U} \neq \emptyset$.

(A1) $\forall\ t\ \mathsf{Reach}_{\mathcal{A}}(t, t + \frac{\epsilon}{2}) \not\subseteq \mathcal{U} \wedge \mathsf{Reach}_{\mathcal{A}}^{T} \cap \mathsf{Shrink}(\mathcal{U}, \gamma) \neq \emptyset \Rightarrow$ *POSSIBLY UNSAFE.*

(B) *UNDECIDED* $\Rightarrow \forall\ t\ \mathsf{Expand}(\mathsf{Reach}_{\mathcal{A}}(t, t + 2\epsilon), \gamma) \not\subseteq \mathcal{U} \wedge \mathsf{Reach}_{\mathcal{A}}^{T} \cap \mathsf{Shrink}(\mathcal{U}, \gamma) = \emptyset$.

Proof. $(A0)$ If Verify2 terminates POSSIBLY UNSAFE, then $\exists \sigma$ that set PossiblyUnsafeFlag to **true** where σ is bounded between $\frac{\epsilon}{2}$ and ϵ such that $C(\sigma) \cap \mathsf{Shrink}(\mathcal{U}, \gamma) \neq \emptyset$. Using Assumption 3.1.1, $C(\sigma) \subseteq \mathsf{Expand}(\mathsf{Reach}_{\mathcal{A}}(\sigma), \gamma)$ and $\mathsf{Expand}(\mathsf{Reach}_{\mathcal{A}}(\sigma), \gamma) \cap \mathsf{Shrink}(\mathcal{U}, \gamma) \neq \emptyset$. This implies $\mathsf{Reach}_{\mathcal{A}}(\sigma) \cap \mathcal{U} \neq \emptyset$.

Also, for all the leaves σ UnsafeFlag was not set to **true**, so we can use the proof from part $(C0)$ of the proof of Proposition 1 to show $\forall\ t$ $\mathsf{Expand}(\mathsf{Reach}_{\mathcal{A}}(t, t + 2\epsilon), \gamma)$.

($A1$) If $\forall\ t$, $\mathsf{Reach}_{\mathcal{A}}(t, t + \frac{\epsilon}{2}) \not\subseteq \mathcal{U}$, then $\forall\sigma$, $C(\sigma) \not\subseteq \mathcal{U}$ and σ does not set UnsafeFlag to **true**. If $\mathsf{Reach}^T_{\mathcal{A}} \cap \mathsf{Shrink}(\mathcal{U}, \gamma) \neq \emptyset$, then $\exists\sigma$ where $C(\sigma) \cap \mathsf{Shrink}(\mathcal{U}, \gamma) \neq \emptyset$ such that PossiblyUnsafeFlag is set to **true** and Verify2 will return POSSIBLY UNSAFE.
(B) Since UNDECIDED, $\forall\sigma$ UnsafeFlag was not set to **true**. The proof is the same as in part ($C0$) of the proof of Proposition 1 to show $\forall t\ \mathsf{Expand}(\mathsf{Reach}_{\mathcal{A}}(t, t+2\epsilon), \gamma) \not\subseteq \mathcal{U}$. Also, $\forall\sigma$ PossiblyUnsafeFlag was not set to **true**. So for all σ, $C(\sigma) \cap \mathsf{Shrink}(\mathcal{U}, \gamma) = \emptyset$, and $\mathsf{Reach}^T_{\mathcal{A}} \subseteq \bigcup_{\forall\sigma} C(\sigma)$.

□

In this chapter we presented a verification algorithm that checks safety properties for systems that are composed of non-interacting HA components. Two versions of the algorithm were presented depending on whether approximate or exact reach sets can be computed. With exact reach sets, more information can be determined from the undecided case to differentiate between UNDECIDED and POSSIBLY UNSAFE. Here, POSSIBLY UNSAFE indicates that at least one, but not all, executions of the actual system enter the unsafe set.

CHAPTER 4

SATELLITE CASE STUDY

In this chapter, we present experimental results on safety verification of a two-satellite system, using the techniques of Chapter 3. For modern satellite systems with significant autonomous capabilities, uncertainties in the model arise from various sources such as uncertain orbit parameters, uncertain initial conditions, or sensor errors. Formal verification techniques can establish safety while accounting for such uncertainties, making them attractive analysis tools. It is also the case that satellite positions are observed periodically from Earth-based stations to confirm their location, and therefore, bounded time safety verification is often sufficient.

We will verify the following property: **conjunction avoidance**: given two passive (non-thrusting) satellites, they do not come closer than a certain distance within the time horizon T. The satellites can be modeled as Hybrid Automaton $\mathcal{A}_1$, $\mathcal{A}_2$ where the composition is $\mathcal{A} = \mathcal{A}_1 \| \mathcal{A}_2$. We will compare two methods for verifying conjunction avoidance: first, by directly computing $\mathsf{Reach}^T_{\mathcal{A}}$, and second, by computing $\mathsf{Reach}^T_{\mathcal{A}_1}$ and $\mathsf{Reach}^T_{\mathcal{A}_2}$ independently and using Algorithm 1. The results from the first method have been reported in [19].

4.1 Astrodynamics Background

A *satellite* is an object moving around the Earth under the influence of the latter's gravitational force. By Kepler's first law, the orbit of a satellite is an ellipse with the Earth at one of the foci, called the *main focus*, and thus the satellite remains in the same plane in 3-dimensional space.[1] Different orbits may or may not be coplanar or coaxial. The masses of the Earth and

[1] Generally, an orbit is some conic section, but we assume orbits are circular or elliptical (the eccentricity e of the orbit satisfies $0 \leq e < 1$).

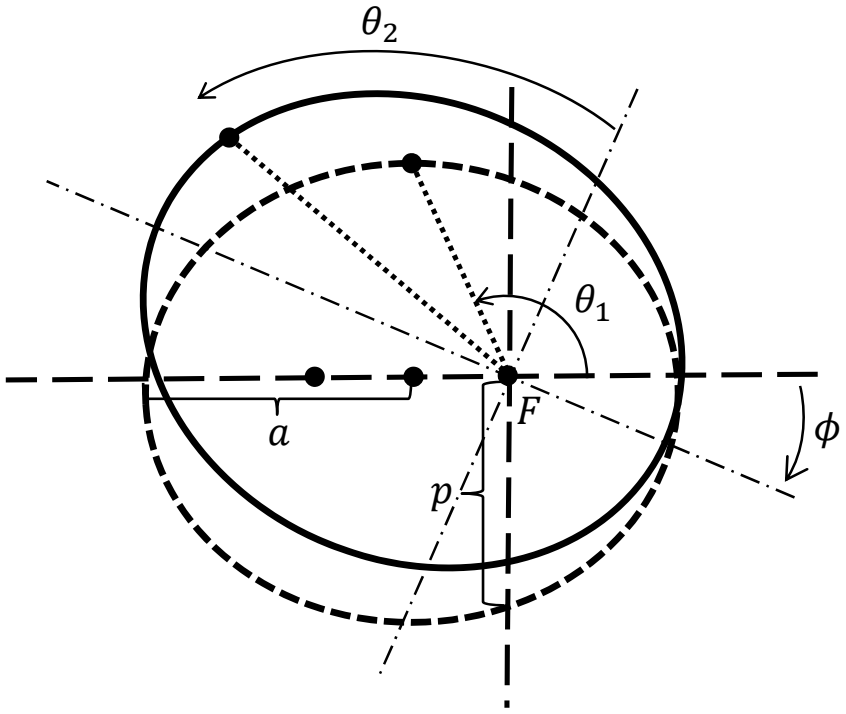

Figure 4.1: Two non-coaxial satellite orbits: θ_1 and θ_2 are the satellite angular positions, a is the semi-major axis, p is the semi-latus rectum, ϕ is the angular offset between the satellite axes, and F is the common foci the satellites are orbiting around.

the satellite, and the velocity of the satellite (with respect to Earth) at a particular position in space, uniquely define the orbit it is on.

Fixing an orbit, a satellite's motion in polar coordinates is given by the following equation:

$$\frac{d}{dt}(\theta) = f(\theta, p, e) = \sqrt{\frac{\mu}{p^3}}(1 + e\cos\theta)^2, \tag{4.1}$$

where θ is the angle of the satellite with respect to the major axis as measured from the main focus (known as the *true anomaly*), e is the *eccentricity*, $p = a(1-e^2)$ is called the *semi-latus rectum*, a is the *semi-major axis*, and μ is the geocentric gravitational parameter. See Fig. 4.1 for a graphical depiction of these quantities. This equation essentially captures Kepler's law of equal

areas. We refer the interested reader to standard texts in astrodynamics (for example [37, 38, 39]) for derivations of this equation. Given an angle θ, Cartesian coordinates of the satellite are specified by

$$r = \frac{p}{1 + e\cos\theta},\ x = r\cos\theta,\ \text{and}\ y = r\sin\theta. \tag{4.2}$$

The conjunction avoidance property is defined in terms of the proximity of the two satellites measured by their Euclidean distance in 3-dimensional space. Given two orbits o_1, o_2, and a distance threshold d, we define the set $P_d(o_1, o_2) \subseteq \mathbb{R}^2$ to be all (θ_1, θ_2) values at which the distance between the orbits is at most d. For verifying conjunction avoidance the unsafe set $\mathcal{U} = P_d(o_1, o_2)$. For coplanar orbits,

$$P_d(o_1, o_2) \triangleq \{(\theta_1, \theta_2)\ :\ ||(x_1, y_1) - (x_2, y_2)|| \leq d\} \tag{4.3}$$

where $||\cdot||$ is the 2-norm. For non-coaxial and non-coplanar orbit pairs, the expression for $P_d(o_1, o_2)$ is analogous, albeit more complex.[2] Although (4.1) models the angular positions θ_1 and θ_2 as unbounded quantities, with respect to the distance predicate P_d, it suffices to look at $\theta_1 \mod 2\pi$ and $\theta_2 \mod 2\pi$. Therefore, in constructing the hybrid automaton model of the satellite we model the angular positions as continuous variables of type $[0, 2\pi]$ and add the appropriate wrap-around transitions.

4.2 Hybridization

Since the software tools for computing the reach set of nonlinear systems are not as well-developed as those for linear and rectangular HA, we abstract the given nonlinear system by a HA with simpler dynamics. We employ the *hybridization* approach developed in [29, 30]. The key idea is to partition the state space $[0, 2\pi]$ into a finite number of zones and then conservatively approximate the nonlinear dynamics in each zone with simpler dynamics. In our case we use rectangular dynamics. The HA model of a single satellite is

[2] Non-coaxial orbits are shown in Fig. 4.1 and are used in the experiments. Non-coplanar orbits require the introduction of more orbital parameters, but we note that all the methods presented in this chapter apply for non-coaxial and non-coplanar orbits as well.

constructed as follows.

Definition 5. *Given a sequence of points* $0 < r_1 < \ldots < r_{k-1} < 2\pi$ *spanning the* $[0, 2\pi]$ *state space the resulting satellite HIOA is a tuple* $\mathcal{A} \triangleq \langle V, L, \Theta, A, \mathcal{D}, \mathcal{T} \rangle$*, where:*

(a) V *is the set of variables.* $X = \{\theta, timer\}$ *and the set of discrete locations* $L = \{l_1 \ldots l_k\}$*. There are no input or output variables, i.e.,* $U = Y = \emptyset$*.*

(b) $\Theta \subseteq Q$*.*

(c) $A = \{next\}$*.*

(d) $(\mathbf{x}, \ell) \overset{next}{\rightarrow} (\mathbf{x}', \ell')$ *iff one of the following conditions hold:*

(i) $\mathbf{x} \lceil \theta = \mathbf{x}' \lceil \theta = r_j$ *for some partition point* r_j*, and* $\ell = l_j$*,* $\ell' = l_{j+1}$*, and* $\mathbf{x} \lceil timer = \mathbf{x}' \lceil timer$*.*

(ii) $\mathbf{x} \lceil \theta = 2\pi$*,* $\mathbf{x}' \lceil \theta = 0$ *and* $\ell = l_k$*,* $\ell' = l_1$*, and* $\mathbf{x} \lceil timer = \mathbf{x}' \lceil timer$*.*

(e) $\mathcal{T}$ *is a set of trajectories for* X *specified by invariants, stopping conditions, and differential inequalities for each* $\ell \in L$*. For each* $\ell \in L$*,*

(i) $Flow(\ell)$*:*

$$\frac{d}{dt}(timer) = 1, \frac{d}{dt}(\theta) \in [a_\ell, b_\ell] \text{ where,}$$
$$a_\ell = \min_{\theta \in [r_{\ell-1}, r_\ell]} f(\theta, p, e),$$
$$b_\ell = \max_{\theta \in [r_{\ell-1}, r_\ell]} f(\theta, p, e).$$

(ii) $Inv(\ell) = [r_{\ell-1}, r_\ell]$

(iii) $Stop(\ell) = (\theta \geq r_\ell)$

The complete two-satellite system is then the composition $\mathcal{A} = \mathcal{A}_1 || \mathcal{A}_2$.

4.3 Experimental Results

The implementation of the verification process has three parts: (i) A *hybridizer* which generates the HA model for the satellites using procedure

of Definition 5, (ii) A *reachability tool* PHAVer [11] which computes the bounded reach sets of the HA produced by (i), (iii) A *verifier* which implements Algorithm 1 for checking conjunction avoidance. For direct verification (monolithic), a single automaton modelling two satellites is verified with PHAVer directly after hybridization. The reach set output from step (ii) is represented as a collection of convex polyhedra. Our implementation of Algorithm 1 analyzes this output using the functions provided in the Parma Polyhedra Library (PPL) [23].

The first set of experiments compared two different methods of safety verification for the satellites. The *monolithic* method computes $\mathsf{Reach}_{\mathcal{A}}^{T}$ for the composite system and then checks the intersection of the reach set with the unsafe distant set $\mathcal{U}$. There are only two safety decisions for this approach, namely, SAFE or M-UNSAFE. If SAFE, then the intersection between $\mathsf{Reach}_{\mathcal{A}}^{T}$ and $\mathcal{U}$ is empty. If M-UNSAFE, then the intersection is non-empty. Unlike the algorithms presented in the previous chapter, in this approach there are no checks to see if the reach set over a particular time interval is a subset of the unsafe set. The *compositional* method first computes Reach_{1}^{T} and Reach_{2}^{T}. Then, since the satellite HA definition is an abstraction of the actual non-linear system, we use Verify1 to check safety where the safety decisions are SAFE, UNSAFE, or UNDECIDED. The semantics of these safety decisions were outlined in Proposition 1. As an example of an experiment that decided SAFE for both monolithic method and compositional method, refer to Fig. 4.2 and Fig. 4.3.

The results of the comparison between these two methods are shown in Table 4.1. The experiments were run on a laptop computer with an Intel i5 processor running a Virtual Machine with Fedora that was allocated 2GB of memory. Each test case in the table has two sub-rows—an M denotes the monolithic approach and a C denotes the compositional approach. We highlight the key observations from these experiments.

Savings in Reachability Computations For the compositional tests, the relative run-time between the reachability computations and Verify1 were comparable. In the monolithic test cases the majority of the run-time for the verification procedure is in the PHAVer run time for generating the reach sets. In the monolithic tests the total run-time is mostly in the reachability computation where the PHAVer runtime is typically around three orders of

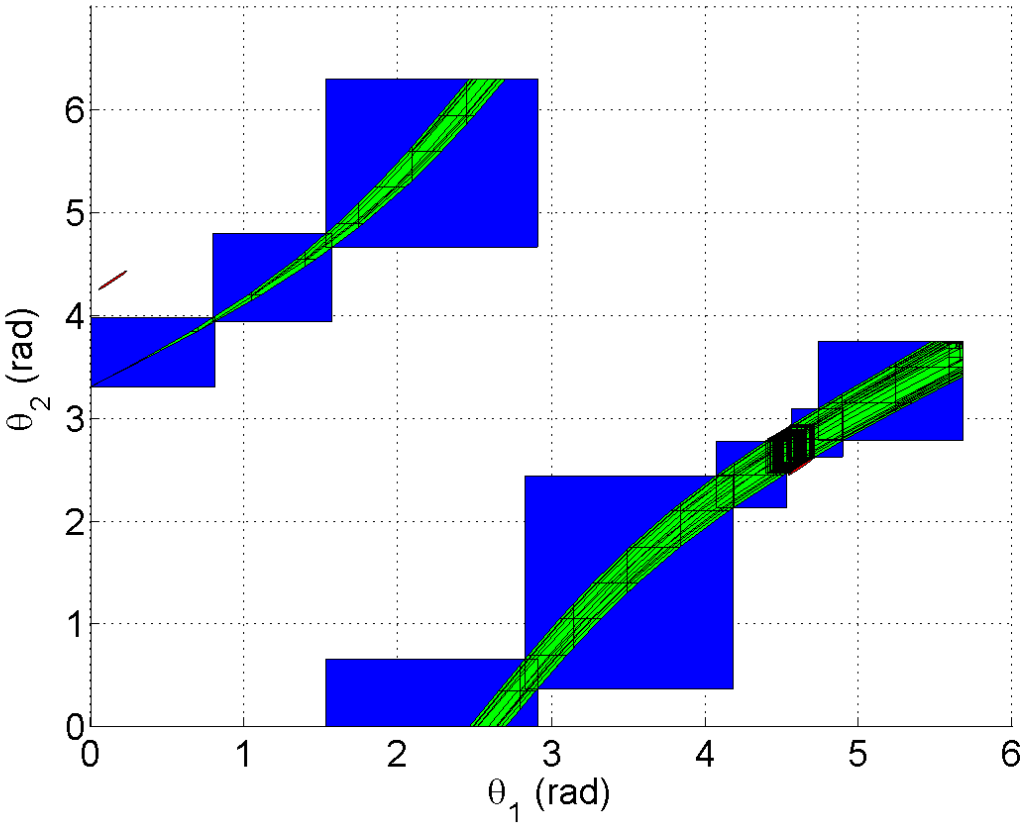

Figure 4.2: From Table 4.1, test case nine where both the monolithic method and compositional method decided safe. Green set is monolithic reach set. For a run of Verify1, the blue set is $\bigcup_{\sigma \in \Lambda} C(\sigma)$. The red set is $\mathcal{U}$, located near $(.25, 4.3)$ and $(4.64, 2.54)$.

magnitude greater than that for the compositional method. In addition, the peak memory usage in PHAVer is around two to three orders of magnitude greater in the monolithic method. These run-time and memory costs are not that surprising because the composed system has three continuous variables while each component automaton has only two. Also, for the individual automata, which may have n discrete locations, the composition used to create the single two satellite automaton has n^2 discrete locations.

Hybridization Precision Test cases three through seven illustrate how verification scales with more precise hybridization. Across these test cases, the orbital parameters, initial condition, and time bound all remain the same, only the hybridization partition size is varied from 30 degrees down to 5 degrees. The relative savings in computation costs between the two methods

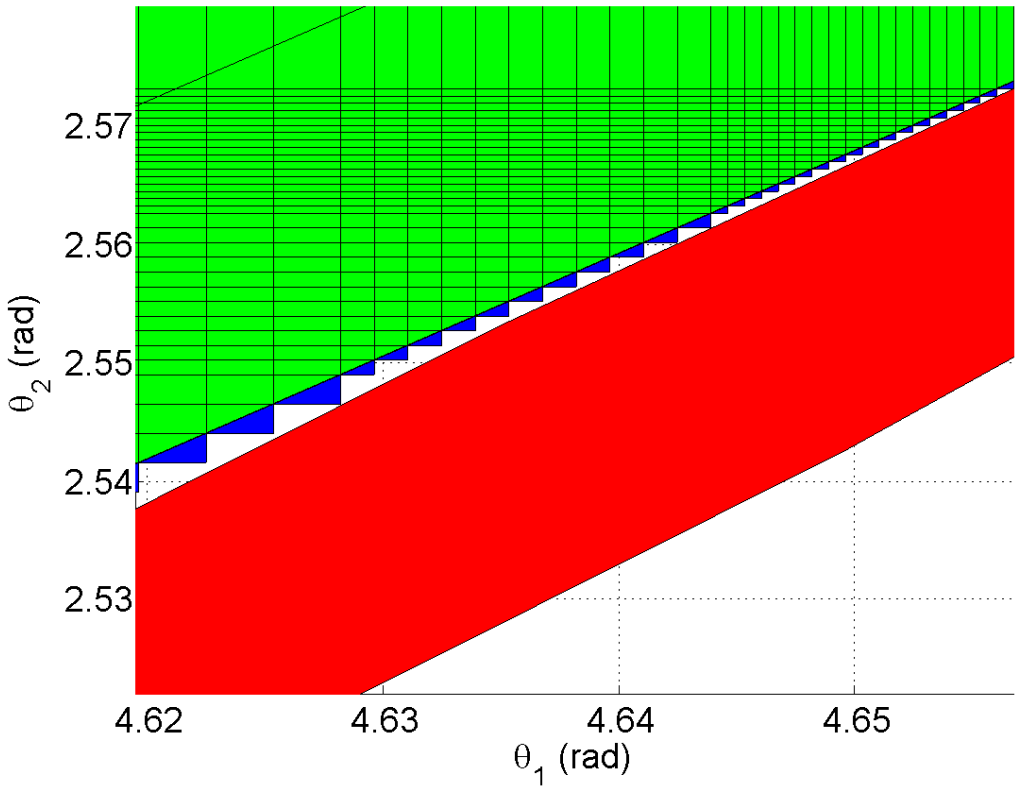

Figure 4.3: Zoomed in region from Fig. 4.2. Green set is monolithic reach set. Blue set is $\bigcup_{\sigma \in \Lambda} C(\sigma)$, and red set is $\mathcal{U}$.

becomes greater the smaller the partition size, showing how the algorithm scales. For these test cases, Fig. 4.4 compares the computation costs between the monolithic and compositional methods.

Shape of Orbits For test cases twelve through fourteen, the computation costs are greater compared to other tests with the same time bound and partition size. This is because these orbits have higher eccentricities which in turn create larger rectangular inclusions in hybridization and larger reach sets with many more polyhedra.

Now, consider the test cases in Table 4.2 which only use the compositional approach with Verify1. These tests use finer partitions in hybridization and produce composed automata which cannot be handled by the monolithic approach. For these smaller partitions, the reach set compositions used in Verify1 are more exact, and it is possible to decide UNSAFE as in Fig. 4.5. Overall, the runtime of these experiments with the smaller partition sizes were significantly higher.

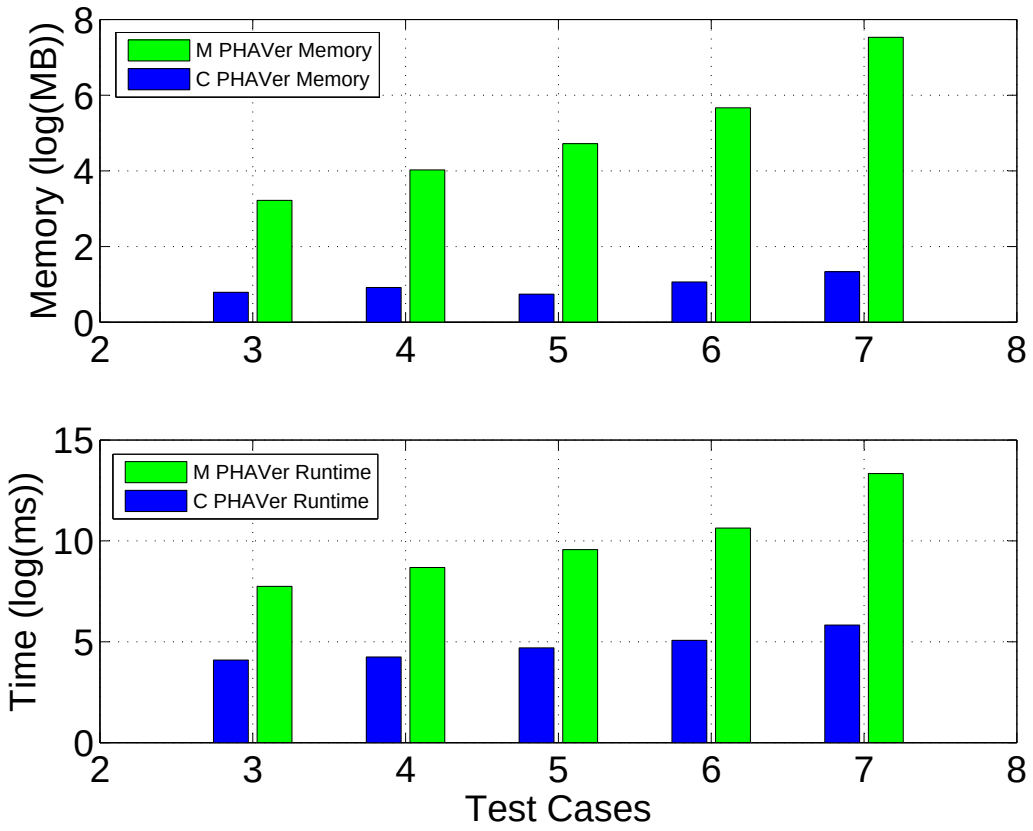

Figure 4.4: Computation costs comparing the monolithic method and the compositional method. Plots are logarithmic, comparing the peak PHAVer memory usage in the top plot and the PHAVer runtimes on the bottom.

Considering test cases seven through nine, the orbit parameters, initial conditions, and time bound all remain the same. The partition size is progressively halved going from 0.1 to 0.05 to 0.025 degrees. The smaller partition resolution allows for more accurate safety decisions to be made, but towards these smaller partition sizes, the computation costs start to become significant in both run time and memory.

In summary, these case studies have shown the effectiveness of using the techniques presented in Chapter 3 for verifying systems loosely coupled through a shared time axis. The satellite systems analyzed here had parameters describing realistic orbits that were verified (in some cases in seconds) for useful durations of time using off the shelf hardware. For example, for a duration of six orbits with one degree partitions, the conjunction avoidance property was established in 41 seconds. Depending on the desired accuracy of the overapproximated reach set, these techniques could be used in real-time verification

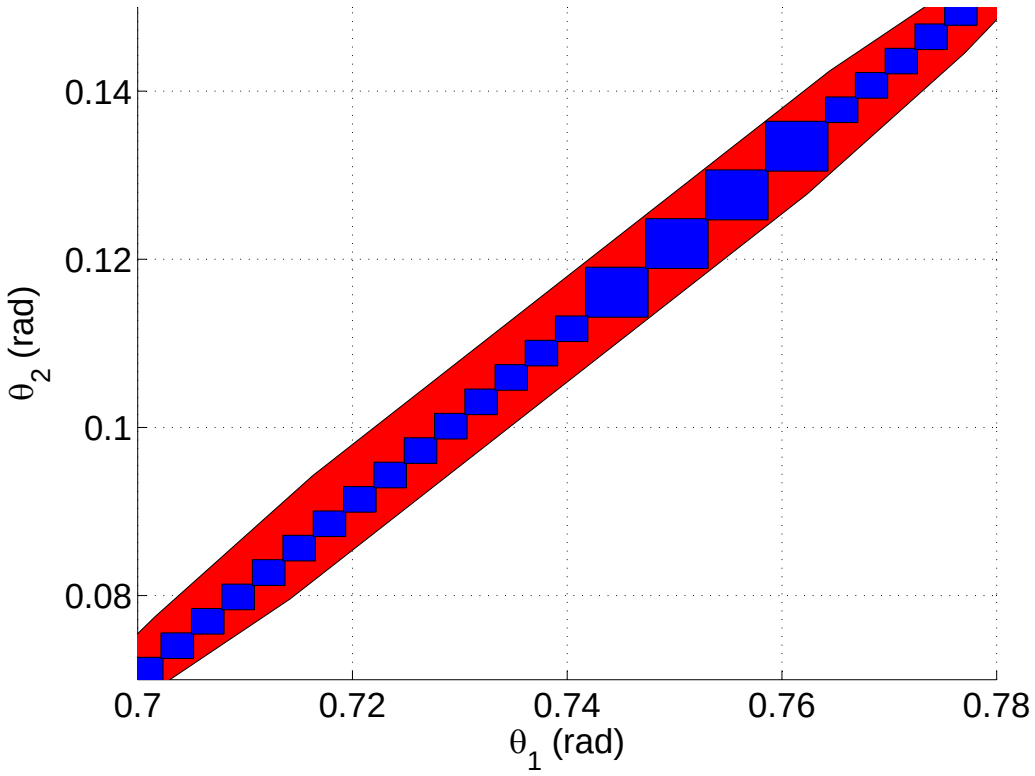

Figure 4.5: From Table 4.2, zoomed in portion of test case seven which decided unsafe. For the run of Verify1, the blue set is $\bigcup_{\sigma\in\Lambda} C(\sigma)$. Several compositions are contained in the red unsafe set $\mathcal{U}$.

as well.

Table 4.1: Summary of results on safety verification for two-satellite conjunction avoidance using two approaches. The first approach models the system as a single automaton (monolithic) and the second models each satellite as a separate automaton (compositional) and then checks safety with Algorithm 1. The *eccentricity* e and *semi-major axis* a are the orbit parameters. The *axis offset* ϕ is the rotation of satellite two's coordinate frame with respect to satellite one. P-T, P-M, V-T, T-T are the phaver run time, phaver peak memory usage, verification run time, and total run time respectively. The unsafe set $\mathcal{U}$ is constructed from a distance threshold of $100km$ between the satellites. The Verify column indicates the number of calls to ReachCompCheck1.

Test		Parameters	Initial	T-bound	Partition	P-T	P-M	V-T	T-T	Verify	Safety
		$(e_1, a_1[km], e_2, a_2[km], \phi[rad])$	(ν_1, ν_2)	orbits	deg	(s)	MB	(s)	(s)	calls	
1	M	$(0.05, 7053, 0.1, 7689, \frac{\pi}{3})$	$(0.3, 0.5)$	2	20	77.16	344	1.4	81.47		SAFE
	C					0.1	2.8	0.27	0.78	75	SAFE
2	M	”	”	1	10	14.27	144	0.26	15.45		SAFE
	C					0.13	2.9	0.03	0.6	7	SAFE
3	M	$(0.07, 7467, 0.15, 6718, \frac{2\pi}{3})$	$(0, 3.3)$	1	30	2.32	25	0.08	2.87		M-UNSAFE
	C					0.06	2.2	1.49	1.99	983	UNDECIDED
4	M	”	”	”	20	5.89	56	0.18	6.76		M-UNSAFE
	C					0.07	2.5	0.54	1.03	231	UNDECIDED
5	M	”	”	”	15	14.33	112	0.32	15.63		M-UNSAFE
	C					0.11	2.1	0.26	0.75	87	UNDECIDED
6	M	”	”	”	10	41.6	289	0.8	44.27		SAFE
	C					0.16	2.9	0.08	0.63	17	SAFE
7	M	”	”	”	5	619.9	1863	4.57	634.7		SAFE

Continued on next page

Table 4.1 – continued from previous page

Test		Parameters	Initial	T-bound	Partition	P-T	P-M	V-T	T-T	Verify	Safety
	C					0.34	3.8	0.15	0.95	15	SAFE
8	M	”	$(0.4, 3.9)$	1	10	35.84	269	0.72	38.2		SAFE
	C					0.14	3.0	0.09	0.66	17	SAFE
9	M	”	$(0, 3.3)$	0.9	20	4.51	46	0.12	5.17		SAFE
	C					0.07	2.2	0.2	1.61	91	SAFE
10	M	$(0.22, 7355, 0.13, 7732, -\frac{\pi}{3})$	$(0.5, 1.4)$	1	10	45.17	299	0.9	47.98		SAFE
	C					0.08	3	0.18	1.21	31	SAFE
11	M	”	$(0.5, 1.5)$	”	”	89.47	459	1.39	93.75		M-UNSAFE
	C					0.13	2.9	1.82	2.34	545	UNDECIDED
12	M	$(0.328, 19024, 0.324, 20184, \frac{\pi}{5})$	$(2.5, 0.5)$	1	20	14.19	107	0.41	17.2		SAFE
	C					0.12	2.5	0.03	2.16	9	SAFE
13	M	”	$(2, 0.5)$	”	10	1078	1936	7.18	1100		M-UNSAFE
	C					0.14	2.9	1.71	3.81	601	UNDECIDED
14	M	$(0.824, 42164, 0.324, 20184, \frac{\pi}{3})$	$(0.25, 0.25)$	0.5	10	255.9	804	2.46	267.8		SAFE
	C					0.14	3.2	0.05	4.54	11	SAFE

Table 4.2: Summary of test cases that verify safety using Algorithm 1. The *eccentricity* e and *semi-major axis* a are the orbit parameters. The *axis offset* ϕ is the rotation of satellite two's coordinate frame with respect to satellite one. P-T, P-M, V-T, T-T are the phaver run time, phaver peak memory usage, verification run time, and total run time respectively. The Distance column is the distance threshold used to construct $\mathcal{U}$. The Verify column indicates the number of calls to ReachCompCheck1.

Test	Parameters $(e_1, a_1[km], e_2, a_2[km], \phi[rad])$	Initial (ν_1, ν_2)	Distance (km)	T-bound orbits	Partition deg	P-T (s)	P-M MB	V-T (s)	T-T (s)	Verify calls	Safety
1	$(0.05, 7053, 0.1, 7689, \frac{2\pi}{3})$	$(2.25, 0.25)$	100	6	1	10.85	64	2.97	41.32	45	SAFE
2	”	”	”	”	0.5	40.01	188	106.05	147.36	45	SAFE
3	”	$(2.5, 1.377)$	100	1	0.1	40.26	138	61.3	102.08	259	UNSAFE
4	”	”	20	”	”	121.49	138	87.7	214.43	159	UNDECIDED
5	$(0.328, 19024, 0.324, 20184, \frac{\pi}{5})$	$(2, 0.75)$	100	1	0.1	38.97	139	13.82	54.55	49	SAFE
6	”	$(0, 5.709)$	100	0.25	0.1	7.04	80	6.02	15.11	107	UNSAFE
7	”	”	100	0.5	0.1	11.87	91	10.63	24.2	109	UNSAFE
8	”	”	50	”	0.05	48.94	206	33.21	89.59	109	UNSAFE
9	”	”	20	”	0.025	203.06	481	121.66	329.61	107	UNSAFE
10	$(0.824, 42164, 0.324, 20184, \frac{\pi}{3})$	$(0.25, 0.25)$	100	1	0.1	125.05	139	82.76	214.76	17	SAFE

CHAPTER 5

DECOMPOSITION ABSTRACTION

In this chapter we show how bounded reach sets of feed-forward networks of HIOAs can be computed by abstracting the continuous interactions. Consider a pair of automata $\mathcal{A}_1, \mathcal{A}_2$ in a feed-forward chain. Say, the output u from $\mathcal{A}_1$ affects the evolution of $\mathcal{A}_2$'s state variable x. First, the effect of u on x's evolution is overapproximated by creating an abstraction $\mathcal{B}$ of $\mathcal{A}_2$ in the context of $\mathcal{A}_1$. By construction, $\mathcal{A}_1$ and $\mathcal{B}$ are coupled only by the shared time. Then, Algorithm 1 can be used as before to determine global safety properties of the composed system.

5.1 Abstraction Definition

Definition 6. *Given HIOA $\mathcal{A}_1$ and $\mathcal{A}_2$ where $\mathcal{A}_2$ has a variable dependency on $\mathcal{A}_1$, a time bound T, and a sequence Γ of $n-1$ time points $0 < t_1 < t_2 < \ldots < t_n - 1 < T$, the* hybrid I/O automaton (HIOA) abstraction *of $\mathcal{A}_2$ is a tuple $\mathcal{B} \triangleq \langle V', L', \Theta', A', \mathcal{D}', \mathcal{T}' \rangle$.*

(a) $V' \triangleq X' \cup Y' \cup U'$, where $X' = X_2 \cup \{timer\}$, $Y' = Y_2$, $U' = U_2$; timer is a real-valued continuous variable.

(b) $L' \triangleq L_2 \times [n]$,

(c) $\Theta' \triangleq \{(s,t) | s \in \Theta, timer = 0\}$,

(d) $A' \triangleq A_2 \cup \{tick\}$,

(e) $((l, \mathbf{x}) \xrightarrow[\mathcal{B}]{a} (l', \mathbf{x}'))$ iff one of the following conditions hold:

(i) $a \neq tick$ and $\mathbf{x}.timer = \mathbf{x}'.timer$ and $(l[1], \mathbf{x} \lceil X_2) \xrightarrow[\mathcal{A}]{a} (l'[1], \mathbf{x}' \lceil X_2)$, or

(ii) $a = tick$ *and* $\mathbf{x}.timer = t_k$*, for some time point,* $l'[2] = l[2] + 1$ *and* $\mathbf{x} \lceil X_2 = \mathbf{x}' \lceil X_2$.

(f) $\mathcal{T}'$ *is a set of trajectories for* $X_{\mathcal{B}}$*: For any* $\tau' \in \mathcal{T}'$ *the following conditions hold:*

(a) For any $t \in \tau_{\mathcal{B}}.dom$*, if* $t_k \leq t \leq t_{k+1}$ *then* $\tau_{\mathcal{B}} \lceil X_2$ *is a solution of the differential inclusion:*

$$\frac{d}{dt}(\mathbf{x}.x) \in f(\mathbf{x}) + g(proj(C_1(t_k, t_{k+1}), u_1), \ldots, proj(C_1(t_k, t_{k+1}), u_j)) \tag{5.1}$$

where each input variable $u_i \in U_2$ *is mapped to some output variable of* $\mathcal{A}_1$.

(b) For every $t \in \tau_{\mathcal{B}}.dom$*,* $\tau_{\mathcal{B}}(t)$ *satisfies the invariant* $Inv(\tau_{\mathcal{B}}(0).l[1])$ *and* $t_l \leq timer \leq t_u$ *where* $l = t_{\tau_{\mathcal{B}}(0).l[2]}$ *and* $u = t_{\tau_{\mathcal{B}}(0).l[2]+1}$

(c) The stopping condition $Stop = (timer \geq t_u$ *where* $u = \tau_{\mathcal{B}}(0).l[2]+1)$

Using the above definition to construct the abstraction, $\mathcal{B}$ will contain n times the number of states of $\mathcal{A}_2$. Each copy of the states of $\mathcal{A}_2$ in $\mathcal{B}$ will correspond to a specific $[t_k, t_{k+1}]$ time interval. The set of trajectories originally evolved according to a set of differential equations for each $x \in X_2$ as $\mathbf{x}.\dot{x} = f(\mathbf{x}) + g(\mathbf{u})$. Now, for a particular time partition $[t_k, t_{k+1}]$, the valuation of each input variable $u_i \in U_2$ coming from $\mathcal{A}_1$ is replaced with the interval, $C_1(t_k, t_{k+1})$ projected onto u_i. Each $C_1(t_k, t_{k+1})$ (see Definition 3) can be computed as the convex hull of $\mathsf{Reach}_{\mathcal{A}_1}(t_k, t_{k+1})$.

5.2 Soundness of Abstraction

Proposition 3. *Consider a composed HIOA* $\mathcal{A} = \mathcal{A}_1 || \mathcal{A}_2$*, such that all input variables of* $\mathcal{A}_2$ *are output variables of* $\mathcal{A}_1$*. Let* $\mathcal{B}$ *be the abstraction of* $\mathcal{A}_2$ *constructed according to Definition 6. Then,* $\mathsf{Execs}_A \subseteq \mathsf{Execs}_{\mathcal{B}} \downarrow (A_2, V_2)$.

Proof. We will establish that the following relation $R \subseteq Q_2 \times Q_{\mathcal{B}}$ is a simulation relation from $\mathcal{A}_2$ to $\mathcal{B}$ in the context of $\mathcal{A}_1$ which is valid up to time T. In more detail, consider states of $\mathbf{q}_2, \mathbf{q}_2' \in Q_2$ which are reachable within time T such that $\mathbf{q}_2$ goes to $\mathbf{q}_2'$ through either a single action or a trajectory

of some duration t. Also consider a state $\mathbf{q}_{\mathcal{B}} \in Q_{\mathcal{B}}$ such that $\mathbf{q}_2 R \mathbf{q}_{\mathcal{B}}$. We will show that there exists $\mathbf{q}'_{\mathcal{B}}$ such that $\mathbf{q}'_2 R \mathbf{q}'_{\mathcal{B}}$ and that $\mathbf{q}_{\mathcal{B}}$ goes to $\mathbf{q}'_{\mathcal{B}}$ through a sequence of actions and trajectories of the same duration.

We define R as $\mathbf{q}_2 R \mathbf{q}_{\mathcal{B}}$ iff $\mathbf{q}_2 = \mathbf{q}_{\mathcal{B}} \downarrow X_2$.

It is easy to check that for every initial state of $\mathcal{A}_2$ there is a related initial state of $\mathcal{B}$. Now, we fix $\mathbf{q}_2, \mathbf{q}'_2 \in Q_2$, $\mathbf{q}_{\mathcal{B}} \in Q_{\mathcal{B}}$ such that $\mathbf{q}_2$ R $\mathbf{q}_{\mathcal{B}}$ and $\mathbf{q}_2 \rightsquigarrow \mathbf{q}'_2$ for some trajectory τ with duration $d = \tau.dom \in [0, T]$. We construct an execution fragment β as follows: $\beta = \omega_0\ tick\ \omega_1\ tick\ \ldots\ \omega_r$, such that (i) $(\omega_1 \downarrow X_2) \frown (\omega_2 \downarrow X_2) \frown \ldots (\omega_k \downarrow X_m) = \tau$, (ii) $(\omega_1.\mathsf{fstate} \lceil\ timer) = \mathbf{q}_{\mathcal{B}} \lceil\ timer$, (iii) $(\omega_m.\mathsf{lstate} \lceil\ timer) = \mathbf{q}'_{\mathcal{B}} \lceil\ timer$, (iv) for each $i \in \{0, \ldots, m\}$, $(\omega_i \downarrow timer)$ grows monotonically at unit rate, and (v) for each i, $\omega_i.\mathsf{lstate} \lceil\ timer$ is one of the time points in the partition Γ. Clearly, $\beta.\mathsf{fstate} = \mathbf{q}_{\mathcal{B}}$ and $\beta.\mathsf{lstate} = \mathbf{q}'_{\mathcal{B}}$. For condition (v) it is also clear that for each i, $\omega_i.\mathsf{lstate} \stackrel{tick}{\rightarrow} \omega_{i+1}.\mathsf{fstate}$ is a valid discrete transition of $\mathcal{B}$ as the pre-state satisfies the guard condition exactly at the time points. It remains to show that for each i, ω_i is a valid trajectory of $\mathcal{B}$. It suffices to show that $\omega_i \downarrow X_2 = \tau \lceil\ \omega_i.dom$ is a solution of the differential inclusion

$$\begin{aligned} \frac{d}{dt}(\tau(t)) \quad \in \quad & f(\tau(t)) + g(proj(C_1(t_i, t_{i+1}), u_1), \ldots, \\ & proj(C_j(t_i, t_{i+1}), u_j)). \end{aligned}$$

over the interval $t \in [t_i, t_{i+1}]$. We know that there exist specific reachable trajectories $\zeta_1, \ldots, \zeta_j$ of the input variables $u_1, \ldots, u_j$ of $\mathcal{A}_2$, and an initial state $\tau(\omega_{i-1}.\mathsf{ltime})$ of $\mathcal{A}_2$ for which $\tau \lceil\ \omega_i.dom$ is a solution of the differential equation

$$\frac{d}{dt}(\tau(t)) = f(\tau(t)) + g(\zeta_1(t), \ldots, \zeta_j(t)).$$

Therefore, it suffices to show that for any $t \in [t_i, t_{i+1}]$, and any of the input variables $u_l \in U_2$ of $\mathcal{A}_2$, and any reachable input trajectory ζ_l, $\zeta_l(t) \in proj(C_1(t_i, t_{i+1}), u_l)$. $C_1(t_i, t_{i+1})$ is constructed from the convex hull of the reachable set of states of $\mathcal{A}_1$ over t_i to t_{i+1}. Since the convex hull of a set is a superset of the original set, $\zeta_1(t)$ will be contained in $proj(C_1(t_i, t_{i+1}), u_l)$. □

5.3 Abstraction Error Bounds

In the remainder of this chapter, we give an upper bound on the quality of the overapproximation that is obtained using Definition 6. In other words, we bound the maximum error between the reachable states of $\mathcal{A}_2$ and the reachable states of its abstraction $\mathcal{B}$. Our main result here establishes that for any desired error bound $\epsilon > 0$, there exists a time partition generating a sequence of $n-1$ timepoints Γ such that the maximum distance between $\mathsf{Reach}^T_{\mathcal{A}_2}$ and $\mathsf{Reach}^T_{\mathcal{B}}$ is ϵ. Although the abstraction $\mathcal{B}$ is hybrid, for the sake of simplicity in this discussion, we restrict $\mathcal{A}_1$ and $\mathcal{A}_2$ to be hybrid automata with one location (without discrete switches). The more general cases including variable dependencies from multiple automata, general feed forward networks of automata, and discrete switches within the non-abstracted automata are discussed at the end of this chapter.

Recall that an output variable of automaton $\mathcal{A}_1$ which acts as an input variable of automaton $\mathcal{A}_2$ is abstracted by a set of timing-based inputs in $\mathcal{B}$ (see Definition 6). Proposition 4 bounds the error between the set of actual input trajectories Π of $\mathcal{A}_2$ and the abstracted set of inputs for $\mathcal{B}$.

For the remainder of this chapter, we fix the time partition Γ to be an equal sized n-partition of $[0, T]$ defined by $n-1$ equidistant time points $0 < t_1 < t_2 < \ldots < t_{n-1} < T$. We will assume that the set of initial states of $\mathcal{A}_1$ is compact, and the functions on the right-hand side of the differential equations defining the evolution of the continuous output variables (input variables of $\mathcal{A}_2$) U_2 are bounded by a constant, say M. These two assumptions imply that there exists a constant $\rho > 0$, such that for any two trajectories τ, τ' of U_2, $\forall t \in [0, T]$, $|\tau(t) - \tau'(t)| \leq \rho$.

Proposition 4. *Let $\eta = \omega_0$ tick ω_1 tick ... tick ω_{n-1} be a $(\{tick\}, V)$ hybrid sequence of duration T, such that (i) $\omega_k.\mathsf{lstate} = \omega_{k+1}.\mathsf{fstate}$, (ii) $\omega_k.dom = [0, t_{k+1} - t_k]$, and (iii) For all $s \in [0, t_{k+1} - t_k]$*

$$\min_{t \in [t_k, t_{k+1}], \tau \in \Pi} \tau(t) \leq \omega_k(s) \leq \max_{t \in [t_k, t_{k+1}],\ \tau \in \Pi} \tau(t). \tag{5.2}$$

. Then,

$$\forall t \in [0, T], \forall \tau \in \Pi, |\tau(t) - \eta(t)| \leq \rho + \frac{MT}{n}. \tag{5.3}$$

Proof. We bound the difference between $\max_{t\in[t_k,t_{k+1}],\ \tau\in\Pi}\tau(t)$ and $\min_{t\in[t_k,t_{k+1}],\ \tau\in\Pi}\tau(t)$. Let τ_{max} be a trajectory which realizes the upper bound in (5.2) for a particular $[t_k, t_{k+1}]$ interval. The difference $\max_{t\in[t_k,t_{k+1}]}\tau_{max}(t) - \min_{t\in[t_k,t_{k+1}]}\tau_{max}(t)$ is bounded by how much the states of τ_{max} can evolve over a $\frac{T}{n}$ interval. Using the general form of the solution to $\tau_{max}(t)$,

$$\tau_{max}(t+\frac{T}{n}) - \tau_{max}(t) = \int_t^{t+\frac{T}{n}} f(\tau_{max}(s))\mathrm{d}s \leq \frac{MT}{n}. \tag{5.4}$$

Let τ_{min} be a trajectory which realizes the lower bound in (5.2). Since $|\tau_{max}(t) - \tau_{min}(t)| \leq \rho$, the difference $\max_{t\in[t_k,t_{k+1}],\ \tau\in\Pi}\tau(t) - \min_{t\in[t_k,t_{k+1}],\ \tau\in\Pi}\tau(t)$ is bounded by $\frac{MT}{n}+\rho$. Since the $\frac{MT}{n}+\rho$ bound also applies to the difference between each ω_k in η and any $\tau\in\Pi$, η also satisfies this bound for all t. □

If $\mathcal{A}_1$ is deterministic and has a single trajectory τ of length T in Π then the constraints on the $\omega_k(s)$ become:

$$\min_{t\in[t_k,t_{k+1}]}\tau(t) \leq \omega_k(s) \leq \max_{t\in[t_k,t_{k+1}]}\tau(t). \tag{5.5}$$

Consequently ρ becomes 0, and we obtain

$$\forall t\in[0,T],\ |\tau(t)-\eta(t)| \leq \frac{MT}{n}. \tag{5.6}$$

Now we proceed to bound the error between $\mathsf{Reach}^T_{\mathcal{A}_2}$ and $\mathsf{Reach}^T_{\mathcal{B}}$ using Theorem 5. Let the set of initial states of $\mathcal{A}_2$ be bounded by a constant γ, that is, for any $\mathbf{x},\mathbf{x}'\in\Theta_2, |\mathbf{x}'-\mathbf{x}|\leq\gamma$. For this part we also assume that f_1+g_1 is bounded by M_1 and that f_2 and g_2 are Lipschitz with constants K_f and K_g. Recall from our earlier assumption about the dynamics and the initial states of $\mathcal{A}_1$, that there exists a constant $\rho>0$ such that for any two valid input trajectories from $\mathcal{A}_1$, $\zeta_1,\zeta_2\in\Pi$, $\forall t\in[0,T]$, $|\zeta_1(t)-\zeta_2(t)|\leq\rho$.

Theorem 5. *Consider any T-time bounded execution ν of $\mathsf{Execs}_{\mathcal{A}}$. Let ν_1 and ν_2 be the restrictions of ν to $\mathcal{A}_1$ and $\mathcal{A}_2$, and let η be the hybrid sequence constructed from ν_2 and Γ according to Proposition 4. For any execution*

$\beta \in \mathsf{Execs}^T_{\mathcal{B}}$, *and any* $t \in [0,T]$,

$$\begin{gathered}(\nu_2 \downarrow X_2)(t) - (\beta \downarrow X_2)(t) \leq \gamma \exp(K_f t) + \\ \frac{K_g M_1 T}{K_f n}(\exp(K_f t) - 1) + \frac{K_g \rho}{K_f}(\exp(K_f t) - 1)\end{gathered} \tag{5.7}$$

Proof. Fixing ν, η and β, for any $t \in [0,T]$, we define the error $e(t) \triangleq |(\nu_2 \downarrow X_2)(t) - (\beta \downarrow X_2)(t)|$. To show this error bound, we first bound the difference between the differentials of $\mathcal{A}_2$ and its abstraction $\mathcal{B}$. With this, we will derive an expression for $e(t)$ that can be bounded using Gronwall's inequality [40]. The executions ν_2 and β satisfy the differential equations:

$$\begin{aligned}\frac{d}{dt}((\nu_2 \downarrow X_2)(t)) &= f_2((\nu_2 \downarrow X_2)(t)) + g_2((\nu_1 \downarrow U_2)(t)) \\ \frac{d}{dt}((\beta \downarrow X_2)(t)) &= f_2((\beta \downarrow X_2)(t)) + g_2(\eta(t)),\end{aligned}$$

where $\eta(t)$ is an input trajectory satisfying the construction in Definition 6. Then:

$$\begin{aligned}|\frac{d}{dt}((\nu_2 \downarrow X_2)(t)) - \frac{d}{dt}((\beta \downarrow X_2)(t))| = \\ |f_2((\nu_2 \downarrow X_2)(t)) + g_2((\nu_1 \downarrow U_2)(t)) - \\ f_2((\beta \downarrow X_2)(t)) - g_2(\eta(t))|.\end{aligned} \tag{5.8}$$

Rearranging terms, using triangle inequality and Lipschitz assumptions we get:

$$\begin{aligned}|\frac{d}{dt}((\nu_2 \downarrow X_2)(t)) - \frac{d}{dt}((\beta \downarrow X_2)(t))| \leq \\ K_f|(\nu_2 \downarrow X_2)(t) - (\beta \downarrow X_2)(t)| + K_g|(\nu_1 \downarrow U_2)(t) - \eta(t)|.\end{aligned} \tag{5.9}$$

Since $\mathcal{A}_1$ contains a single location without discrete switches, $\nu_1 \downarrow U_2$ is a single trajectory and we can use Proposition 4 to bound the difference between $\nu_1 \downarrow U_2$ and η. We also recognize that $|\frac{d}{dt}((\nu_2 \downarrow X_2)(t)) - \frac{d}{dt}((\beta \downarrow X_2)(t))|$ and $|(\nu_2 \downarrow X_2)(t) - (\beta \downarrow X_2)(t)|$ can be rewritten as $\frac{d}{dt}(e(t))$ and $e(t)$, respectively. Thus, the error along these executions satisfies:

$$\frac{d}{dt}(e(t)) \leq K_f e(t) + K_g(\frac{M_1 T}{n} + \rho). \tag{5.10}$$

From the assumption that the difference between any two initial conditions of $\mathcal{A}_2$ are bounded by γ, we have $\forall \nu_2 \in \mathsf{Execs}^T_{\mathcal{A}_2}$, $\beta \in \mathsf{Execs}^T_{\mathcal{B}}$, $|\nu_2(0) - \beta(0)| \leq \gamma$, and $e(0) \leq \gamma$ such that the general solution of $e(t)$ is:

$$\begin{aligned} e(t) &\leq e(0) + \int_0^t K_f e(s) + K_g(\frac{M_1 T}{n} + \rho)\mathrm{d}s \\ e(t) &\leq \gamma + \frac{K_g M_1 T}{n} t + K_g \rho t + \int_0^t K_f e(s)\mathrm{d}s \end{aligned} \tag{5.11}$$

Applying the Gronwall-Bellman inequality:

$$\begin{aligned} e(t) \leq \gamma + \frac{K_g M_1 T}{n} t + K_g \rho t + \\ K_f \int_0^t \left(\left(\gamma + \left(\frac{K_g M_1 T}{n} + K_g \rho \right) s \right) * \exp \int_s^t K_f \mathrm{d}r \right) \mathrm{d}s. \end{aligned} \tag{5.12}$$

Carrying out the integration yields the desired bound:

$$\begin{aligned} e(t) \leq \frac{K_g M_1 T}{K_f n} \left(\exp\left(K_f t \right) - 1 \right) + \\ \frac{K_g \rho}{K_f} \left(\exp\left(K_f t \right) - 1 \right) + \gamma \exp\left(K_f t \right). \end{aligned} \tag{5.13}$$

□

Corollary 6. *If the set of start states for $\mathcal{A}_2$ is a singleton and there is a unique input trajectory from $\mathcal{A}_1$, then $\forall \epsilon > 0$, $\exists n$ such that $\forall t \in [0, T]$, $e(t) \leq \epsilon$.*

Proof. Consider the bound on the error in (5.7). Since there is a single initial condition for $\mathcal{A}_2$, $\gamma = 0$, and since there is a single input trajectory from $\mathcal{A}_1$, $|(\nu_1 \downarrow U_2)(t) - \eta(t)|$ is bounded using (5.6) and $\rho = 0$. The error bound becomes:

$$e(t) \leq \frac{K_g M_1 T}{K_f n} \left(\exp\left(K_f t \right) - 1 \right). \tag{5.14}$$

With n in the denominator, we create $\mathcal{B}$ with a sufficiently fine time partition (sufficiently large n) that achieves any error bound ϵ. □

5.4 Generalizations

For the extension of having automata with multiple locations and discrete switches, the above analysis can still apply if all switches between locations have identity resets and the right-hand sides of the differentials are piecewise continuous between locations. In this case, we choose the maximum Lipschitz constants, K_f and K_g, over all locations of a particular automaton. If the automaton supplying the input in the abstraction has multiple locations, then we use the maximum bound M on the differentials describing the evolution of the input signal. In the most general case, with discrete jumps having non-identity resets, the abstraction in Definition 6 can still be applied. However, the conditions for when a discrete jump occurs could be dependent on the abstracted input signal, so a discrete jump in the abstraction could occur before or after times when the discrete jump of $\mathcal{A}_2$ can occur. The error between the continuous states of $\mathcal{A}_2$ and $\mathcal{B}$ at these times will be at least the difference between the guard and the non-identity reset, and thus, an error bound ϵ less than this difference cannot be realized around times when these jumps occur. For variable dependencies from multiple automata, as long as the differentials of the input signals are bounded, similar results can be obtained. For general feed forward networks, let the dependencies between automata in the network be directed edges such that the network forms a directed graph. In this case, each automaton with at least one dependency will require the abstraction to generate its reach set. The general network must be a directed acyclic graph, where an automaton with dependencies is only abstracted once all automata in its dependencies have been previously abstracted or don't have any dependencies themselves.

CHAPTER 6

COMPARTMENTAL ANALYSIS CASE STUDY

6.1 Tank System Overview

As a demonstration of the abstraction presented in Chapter 5, we consider a system composed of a sequence of tanks. The tanks hold some fluid flows from one tank to the next at a constant rate of $r\frac{Liters}{min}$. Such tank systems are commonly used in chemical plants, where the tank sequence is used to mix different reactants.

Let this particular system be a sequence of n brine tanks. Fresh water flows into tank 1, while the mixed brine flows from tank 1 to tank 2, tank 2 to tank 3, and so on. The continuous variable x_i for each tank represents the amount of salt (in kilograms) for that tank. The volume of each tank (Liters) is labeled V_i for each tank. The complete system modeling the amount of salt in the tanks can be viewed as a set of linear differential equations.

$$\begin{aligned}
\frac{d}{dt}(x_1) &= -k_1 x_1, \\
\frac{d}{dt}(x_2) &= k_1 x_1 - k_2 x_2, \\
\frac{d}{dt}(x_3) &= k_2 x_2 - k_3 x_3 \\
&\cdots \\
\frac{d}{dt}(x_n) &= k_{n-1} x_{n-1} - k_n x_n \quad where, \\
k_i &= \frac{r}{V_i}, \quad i \in \{1, 2, \ldots, n\}
\end{aligned}$$

For the first equation that models the first tank in the sequence, the instantaneous rate at which the total salt decays is directly proportional to the flow rate, and it is inversely proportional to the volume of the tank. The later

tanks in the sequence have the mixed brine flowing in from the previous tank, so the total salt may increase for a time before beginning to decay.

To test the abstraction, we will construct an automaton for each tank in the sequence for the compositional method. Each automaton will then have two continuous variables. The first is the continuous variable representing the amount of salt for that tank, and the second is a continuous timer. The automaton for the first tank has no input variable, but for the i^{th} tank where $i > 1$, the input variable is the continuous variable x_{i-1} from the previous tank. For the monolithic model, we will have a single automaton with $n + 1$ continuous variables for the salt in each of the n tanks plus the variable for the timer.

6.2 Experimental Results

Some preliminary experiments have been run to demonstrate the feasibility of using the abstraction and compare the monolithic method with the compositional method. The results are presented in Table 6.1. These tests measured the computational costs in PHAVer for creating the reach sets as well as measuring the abstraction time for the compositional tests. When specifying affine dynamics in PHAVer, the partition size for the on-the-fly partitions that PHAVer uses must be specified. The smaller this parameter, the more accurate the reachable set of states that is computed. For comparing the monolithic and compositional methods it is important to note that when keeping the on-the-fly partitioning equal between methods, the reachable set of states computed using the compositional method will be a subset of the reachable states computed using the monolithic method. This is demonstrated in Fig. 6.1.

Test cases such as test number four in Table 6.1 showed significant improvement over runtime and memory costs of PHAVer when compared with the monolithic approach. However there are some important observations to be noted as follows.

Cost of Time Partitioning Consider test cases eight, nine, and ten. The number of tanks, time bound, and PHAVer partitioning all remain constant while the number of time partitions used in the abstraction is varied. Here it

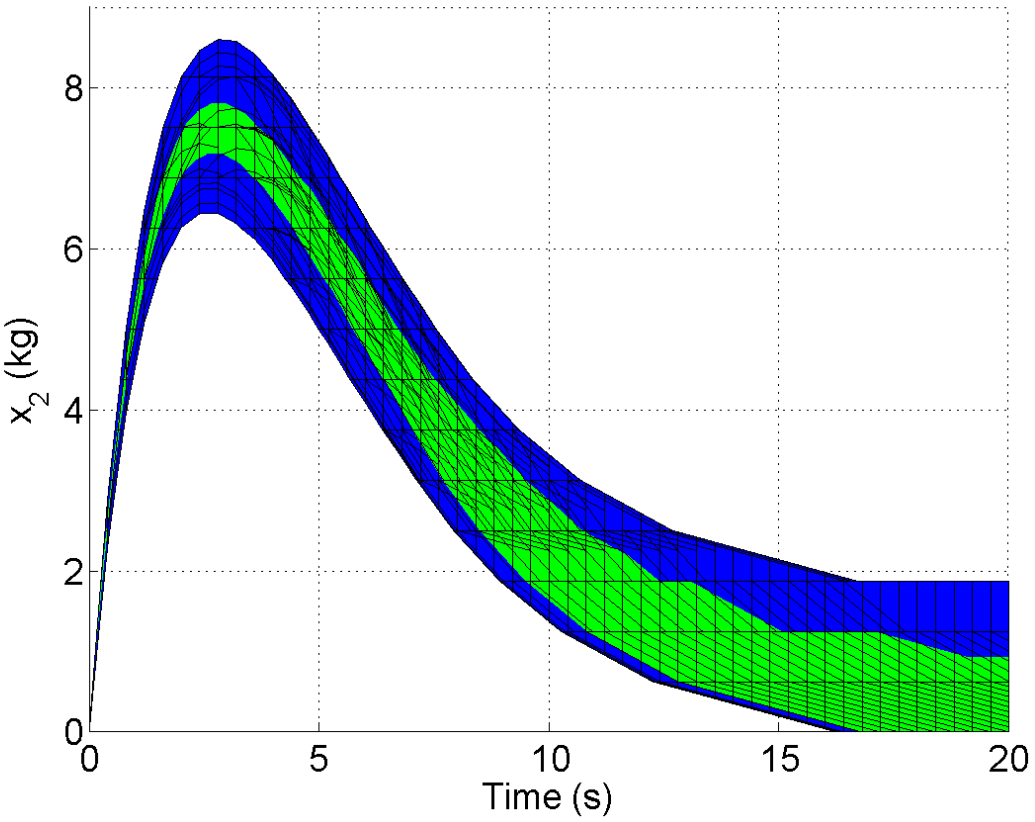

Figure 6.1: From Table 6.1, this is test case four. The green set is the complete reach set from the monolithic test projected onto x_2 and t. The blue set is the reach set generated from the abstracted automaton that modeled the second tank in the system.

can be seen, as expected, that increasing the number of time partitions and hence the number of discrete locations in the abstracted automaton can have a significant effect on the required PHAVer run time and memory usage.

Inefficiency of Abstraction For test cases 11, 12, and 13, where the number of tanks is three or four, the abstraction time outside of PHAVer becomes significant for the compositional method. For this method to become more effective, the techniques used in the abstraction process need to become more efficient.

Despite these issues, when considering the exponential costs with respect to the number of continuous variables in the monolithic approach, the abstraction remains an interesting decompositional approach for future work and improvements.

Table 6.1: These experiments present preliminary results for implementation of the abstraction in Definition 6 for the compartmental analysis example. Time Partitions is the number of time points plus one in the the $[0, T]$ interval for the abstraction. PHAVer partitioning is the size of the on-the-fly partitioning for the continuous variable x_i of each tank. M P-T, M P-M, C P-T, C P-M, and A-T are the monolithic PHAVer time, monolithic PHAVer memory, compositional PHAVer time, compositional PHAVer memory, and abstraction time respectively. All the initial conditions are 0 except for x_1 which equals 15 at $t = 0$. Parameters: $V_1 = 20$, $V_2 = 40$, $V_3 = 50$, $V_4 = 30$, $r = 10$.

Test	# Tanks	T-bound (s)	Time Partitions	PHAVer Partitioning Kg	M P-T (s)	M P-M MB	C P-T (s)	C P-M MB	A-T (s)
1	2	20	50	4	0.11	6.5	0.94	23	0.01
2	”	”	”	3	0.12	6.5	0.92	23	0.03
3	”	”	”	2	0.46	15	1.18	23	0.02
4	”	”	”	1	6.26	147	1.69	28	0.06
5	2	40	50	4	0.12	6.5	0.84	15	0.02
6	”	”	”	3	0.12	6.5	0.83	15	0.02
7	”	”	”	2	0.47	15	1.07	19	0.02
8	2	40	100	2	0.48	15	2.3	38	0.06
9	”	”	200	”	0.48	15	5.84	94	0.13
10	”	”	400	”	0.48	15	18.9	277	0.48
11	3	30	100	4	2.8	59	4.87	58	4.94
12	”	”	”	3	2.9	59	6.23	58	5.19
13	4	20	100	5	5.59	18	25.16	52	102.25

CHAPTER 7

CONCLUSIONS

This thesis presented verification algorithms for systems that can be decomposed into noninteracting components that share the same time axis. For systems whose components form feedforward networks of automata via shared continuous variables, an abstraction is presented that overapproximates their interaction such that the verification algorithm can be used on the abstracted system.

Chapter 2 developed the hybrid automaton model that was used throughout the thesis. In Chapter 3 we presented the algorithms that verify bounded safety properties using the component reach sets of the decomposed system. The main algorithm can give three answers: "Safe", "Unsafe" , and "Undecided". For each of these answers we show sufficient conditions for the complete reach set that will ensure that particular answer. Likewise, given a particular answer, we outline what properties the complete reach set is guaranteed to have. Using a case study on two-satellite systems in Chapter 4, we verified the conjunction avoidance property using realistic orbit parameters. There were two to three orders of magnitude savings in both total runtime and peak memory usage using this approach with component reach sets. In Chapter 5 we considered networks of component automata, where automata are dependent on one another if they have shared continuous variables. We presented an abstraction that overapproximates the interaction between the automata so that the abstractions are only coupled through the shared time variable as in Chapter 3. We established a bound on the error between the component automata of the original system and their abstractions. The feasibility of this approach was demonstrated through experiments that computed this abstraction in Chapter 6.

This suggests several interesting directions for future work.

Efficiency of Abstraction One of the main drawbacks of the current abstraction in Chapter 5 is that it introduces too many additional discrete locations into the abstracted automaton. The number of discrete locations has a significant impact on the cost of the reachability computations. Improving the efficiency of the abstraction will give a better gauge of its effectiveness in comparison to the monolithic approach.

HA generalizations Since the abstraction was used on case studies of continuous dynamical systems, additional analysis and extensions will be required to apply these techniques to hybrid and switched dynamical systems with resets.

New Abstractions In this thesis, our abstraction involved time partitioning where the coupling was overapproximated by rectangular dynamics over each interval of the time partition. It would be interesting to try linear or affine approximations within each time interval. In turn, this may reduce the number of time points necessary in the abstraction to achieve a certain error bound.

Feedback Our work here only considered feedforward networks of HA. An interesting direction would be to develop a new approach that creates a useful abstraction for systems whose components have interactions with feedback.

REFERENCES

[1] M. Williams, "Toyota to recall prius hybrids over abs software," 2010. [Online]. Available: http://www.computerworld.com/s/article/9153938/Toyota_to_recall_Prius_hybrids_over_ABS_software

[2] "Satellite collision leaves significant debris clouds," *Orbital Debris Quarterly News*, vol. 13, no. 2, 2009.

[3] T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya, "What's decidable about hybrid automata?" in *ACM Symposium on Theory of Computing*, 1995, pp. 373–382.

[4] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine, "The algorithmic analysis of hybrid systems," *Theoretical Computer Science*, vol. 138, no. 1, pp. 3–34, 1995.

[5] G. Lafferriere, G. J. Pappas, and S. Sastry, "O-minimal hybrid systems," *Mathematics of Control, Signals, and Systems (MCSS)*, vol. 13, pp. 1–21, 2000.

[6] C. Guernic and A. Girard, "Reachability analysis of hybrid systems using support functions," in *Proceedings of the 21st International Conference on Computer Aided Verification*, ser. Computer Aided Verification. Springer-Verlag, 2009, pp. 540–554.

[7] A. B. Kurzhanski and P. Varaiya, "Ellipsoidal techniques for reachability analysis," in *Proceedings of the Third International Workshop on Hybrid Systems: Computation and Control*, ser. Hybrid Systems: Computation and Control. London, UK, UK: Springer-Verlag, 2000, pp. 202–214.

[8] Z. Han and B. H. Krogh, "Reachability analysis of large-scale affine systems using low-dimensional polytopes," in *Proceedings of the 9th International Conference on Hybrid Systems: Computation and Control*, ser. Hybrid Systems: Computation and Control. Springer-Verlag, 2006, pp. 287–301.

[9] K.-D. Kim, S. Mitra, and P. R. Kumar, "Computing bounded -reach set with finite precision computations for a class of linear hybrid automata," in *Proceedings of the 14th International Conference on Hybrid Systems: Computation and Control*, 2011, pp. 113–122.

[10] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi, "Hytech: A model checker for hybrid systems," *International Journal on Software Tools for Technology Transfer (STTT)*, vol. 1, pp. 110–122, 1997.

[11] G. Frehse, "PHAVer: Algorithmic verification of hybrid systems past hytech," in *Proceedings of the 8th International Conference on Hybrid Systems: Computation and Control*, 2005, pp. 258–273.

[12] J. Bengtsson, K. Larsen, F. Larsson, P. Pettersson, and W. Yi, "Uppaal: a tool suite for automatic verification of real-time systems," in *Hybrid Systems III*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 1996, vol. 1066, pp. 232–243.

[13] E. Asarin, O. Bournez, T. Dang, and O. Maler, "Approximate reachability analysis of piecewise-linear dynamical systems," in *Proceedings of the 3rd International Conference on Hybrid Systems: Computation and Control*, ser. Lecture Notes in Computer Science, N. Lynch and B. Krogh, Eds. Springer Berlin / Heidelberg, 2000, vol. 1790, pp. 20–31.

[14] A. Donzé, "Breach, a toolbox for verification and parameter synthesis of hybrid systems," in *Proceedings of the 22nd International Conference on Computer Aided Verification*, ser. Lecture Notes in Computer Science, T. Touili, B. Cook, and P. Jackson, Eds. Springer Berlin / Heidelberg, 2010, vol. 6174, pp. 167–170.

[15] A. Platzer and J.-D. Quesel, "Keymaera: A hybrid theorem prover for hybrid systems," in *Lecture Notes in Computer Science*. Springer, 2008, pp. 171–178.

[16] S. M. Loos, A. Platzer, and L. Nistor, "Adaptive cruise control: Hybrid, distributed, and now formally verified," in *Formal Methods*, ser. LNCS, M. Butler and W. Schulte, Eds. Springer, 2011.

[17] S. Mitra, Y. Wang, N. A. Lynch, and E. Feron, "Safety verification of model helicopter controller using hybrid input/output automata," in *Proceedings of the 6th International Conference on Hybrid Systems: Computation and Control*, 2003, pp. 343–358.

[18] T. Johnson and S. Mitra, "Parameterized verification of distributed cyber-physical systems:an aircraft landing protocol case study," in *ACM/IEEE Third International Conference on Cyber-Physical Systems, April 2012, Beijing, China*, 2012.

[19] T. T. Johnson, J. Green, S. Mitra, R. Dudley, and R. S. Erwin, "Satellite rendezvous and conjunction avoidance: Case studies in verification of nonlinear hybrid systems," in *Formal Methods*, 2012 (to appear).

[20] T. Wongpiromsarn, S. Mitra, R. Murray, and A. Lamperski, "Periodically controlled hybrid systems: Verifying a controller for an autonomous vehicle," in *Proceedings of the 12th International Conference onHybrid Systems: Computation and Control*, 2009, pp. 396–410.

[21] R. Alur and D. L. Dill, "A theory of timed automata," *Theoretical Computer Science*, vol. 126, pp. 183–235, 1994.

[22] V. Vladimerou, P. Prabhakar, M. Viswanathan, and G. Dullerud, "Stormed hybrid systems," in *Proceedings of the 35th International Colloquium on Automata, Languages and Programming, Part II*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 136–147.

[23] R. Bagnara, P. M. Hill, and E. Zaffanella, "The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems," *Science of Computer Programming*, vol. 72, no. 1–2, pp. 3–21, 2008.

[24] A. Girard, "Reachability of uncertain linear systems using zonotopes," in *Proceedings of the 8th International Conference onHybrid Systems: Computation and Control*, ser. Lecture Notes in Computer Science, M. Morari and L. Thiele, Eds. Springer Berlin / Heidelberg, 2005, vol. 3414, pp. 291–305.

[25] A. A. Kurzhanskiy and P. Varaiya, "Ellipsoidal techniques for reachability analysis of discrete-time linear systems," *IEEE Transactions on Automatic Control*, vol. 52, pp. 26–38, 2007.

[26] P. Prabhakar, V. Vladimerou, M. Viswanathan, and G. E. Dullerud, "Verifying tolerant systems using polynomial approximations," in *Proceedings of the 2009 30th IEEE Real-Time Systems Symposium*, ser. Real Time Systems Symposium. IEEE Computer Society, 2009, pp. 181–190.

[27] A. Chutinan and B. Krogh, "Computational techniques for hybrid system verification," *IEEE Transactions on Automatic Control*, vol. 48, no. 1, pp. 64–75, Jan. 2003.

[28] G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler, "Spaceex: Scalable verification of hybrid systems," in *Proceedings of the 23rd International Conference on Computer Aided Verification*, ser. LNCS. Springer, 2011.

[29] E. Asarin, T. Dang, and A. Girard, "Hybridization methods for the analysis of nonlinear systems," *Acta Informatica*, vol. 43, pp. 451–476, 2007.

[30] T. Dang, O. Maler, and R. Testylier, "Accurate hybridization of nonlinear systems," in *Proceedings of the 13th International Conference on Hybrid Systems: Computation and Control*. ACM, 2010, pp. 11–20.

[31] I. M. Mitchell, "Scalable calculation of reach sets and tubes for nonlinear systems with terminal integrators: A mixed implicit explicit formulation," in *Proceedings of the 14th International Conference on Hybrid Systems: Computation and Control*. ACM, 2011, pp. 103–112.

[32] Z. Han, "Reachability analysis of continuous dynamic systems using dimension reduction and decomposition," Ph.D. dissertation, Carnegie Mellon University, Pittsburgh, PA, USA, 2005, aat 3199803.

[33] K. Shahab and M. Oishi, "Overapproximating the reachable sets of lti systems through a similarity transformation." American Control Conference, June 2010.

[34] N. Lynch, R. Segala, and F. Vaandrager, "Hybrid I/O automata," in *Hybrid Systems III*, ser. Lecture Notes in Computer Science, vol. 1066. Springer-Verlag, 1996, pp. 496–510.

[35] S. Mitra, "A verification framework for hybrid systems," Ph.D. dissertation, Massachusetts Institute of Technology, Cambridge, MA, USA, 2007.

[36] T. A. Henzinger and R. Majumdar, "Symbolic model checking for rectangular hybrid systems," in *Proceedings of the Sixth International Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2000)*. LNCS 1785, Springer-Verlag, 2000, pp. 142–156.

[37] D. Vallado and W. McClain, *Fundamentals of Astrodynamics and Applications*, ser. Space technology library. Microcosm Press, 2001.

[38] R. Battin, *An Introduction to the Mathematics and Methods of Astrodynamics*, ser. AIAA education series. American Institute of Aeronautics and Astronautics, 1999.

[39] R. R. Bate, D. D. Mueller, and J. E. White, *Fundamentals of Astrodynamics*. Dover Publications, 1971.

[40] H. K. Khalil, *Nonlinear Systems*, 3rd ed. Upper Saddle River, NJ: Prentice Hall, 2002.